Automated Handling of Flexible Materials

by

John Charles Briggs

B.S., Massachusetts Institute of Technology
(1986)

Submitted to the Department of
Mechanical Engineering
in Partial Fulfillment of
the Requirements for the Degree of
Master of Science in
Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 1988

Signature of Author_____
Department of Mechanical Engineering
February 10, 1988

Certified by_____
Ming Kai Tse
Thesis Supervisor

Accepted by_____
Ain Ants Sonin
Chairman, Department Committee

Exhibit C

ENGINEERING INTERNSHIP PROGRAM
SCHOOL OF ENGINEERING
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge, Mass. 02139

J.R. MARTUCCELLI

DEC 17 1987

cc M.E Grad Office
REF TO
FILE Student file

THESIS REVIEW LETTER

JRM 12-17-87

Attention: John R. Martuccelli, Director, Room 1-211

Subject: Master's Thesis of John C. Briggs
(student name)

The attached thesis, Automated Handling of Flexible Materials
(Title)

has been reviewed by the undersigned Draper Laboratory
(company name)

representatives and confirmed that it does not contain details objectionable from the standpoint of Draper Laboratory
(company name)

In addition, we understand that the aforementioned thesis report becomes the permanent property of M.I.T., will be placed in the M.I.T. Library within one month of the date of submission and may not be published wholly or in part except by authorization of the Department of Mechanical Engineering
(department name)

in which the student is enrolled. (It is understood that authorization is granted to the Company for such limited publication as the Company's prior contractual obligations may require.)

Draper Laboratory
Company Name

Direct Company Supervisor of Student

Student Name

Approved By (for company)

Title:

Date:

Date: 12/11/87

Date: 12/11/87

Automated Handling of Flexible Materials

by

John Charles Briggs

Submitted to the Department of Mechanical Engineering
on February 19, 1988 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Mechanical Engineering

Abstract

Apparel manufacturers have turned to automation as a means to reduce manufacturing costs. Several recent developments, such as the fully automated suit sleeve assembly line at Draper Labs, are showing promise to fulfill this goal. This assembly line uses fairly complex hardware, including robot folders and vision systems to replace existing manual processes. Unfortunately, the planning and execution of picking up and folding garment pieces remains very complicated. Much expertise and trial-and-error are still needed to determine the folding sequences.

One of the major problems with robot folding is to determine how to pick up pieces of cloth with the robot's end-effector. The end-effector is composed of a multiplicity of single point pick-up devices called pickers. This thesis presents the framework of an expert system for determining suitable sets of picker locations. The problem is that if the pickers are not picking up the cloth at suitable locations, the cloth will distort and get folded under itself. The expert system uses rules to determine picker placement parameters. The development of these rules is based on analytical models and empirical methods. Two of the key rules are the inter-picker spacing rule and the picker-to-edge rule. The inter-picker spacing rule relates cloth shear stiffness to the distance between adjacent pickers. The picker-to-edge rule relates the fabric bending stiffness and the coefficient of friction of the folding surface to the maximum distance between any picker and the cloth edge.

The development and solution of an elastica cloth model was vital to the development of the picker-to-edge rule. The model consists of a heavy elastica contacting a frictional surface. The solution was found on an IBM PS/2 Model 50 using Runge-Kutta integration and the shooting method of solving a two-point boundary value problem. The solution revealed three possible modes of behavior, depending on the coefficient of friction of the folding surface and a dimensionless heavy elastica parameter.

Thesis Supervisor: Dr. Ming Kai Tse
Title: Assistant Professor of
Mechanical Engineering

Acknowledgment

This thesis work was performed at The Charles Stark Draper Laboratory in Cambridge, MA as part of an automated textile equipment project under the sponsorship of Textile Clothing Technology Corporation $(TC)^2$. Their support is much appreciated

I owe a great deal to the Program Manager, William Toth, for his support throughout the year and a half that it took to write this thesis. I would also like to thank all the people at the lab that I consulted with on a day-to-day basis, particularly Peter Wender, Philip Wiener, William Henrickson, Charles Elder, David Barrett, Edward Bernardon, Michael Foley, Ann Ito, and Edward Lane.

My utmost appreciation goes out to Professor Ming Kai Tse, for being my advisor on this thesis. His support and guidance has helped shape my future as a mechanical engineer.

Last, but certainly not least, I would like to thank my wife Sue Ellen. Without her, I don't think I could have endured my long stay at M.I.T. Her encouragement and humor helped to strengthen me against a sometimes heartless institution. This thesis is dedicated to Sue Ellen.

Table of Contents

Abstract 2

Acknowledgment 3

Table of Contents 4

Table of Figures 7

1 Automated Apparel Manufacturing and (TC)^2 11

1.1 Introduction 11

1.2 The Road to Automation 13

1.3 Draper Labs' Automated Sleeve Machine 17

1.4 Expert System for Folding 27

1.5 Expert System for Picker Locations 29

1.6 Overview 35

2 Automated Apparel Manufacturing - Basic Hardware 37

2.1 Sensors for Flexible Materials 37

2.1.1 Introduction 37

2.1.2 Photoelectric Sensors 38

2.1.3 Pneumatic Sensors 44

2.1.4 Ultrasonic Sensors 53

2.1.5 Capacitive Sensors 55

2.1.6 Inductive Proximity Sensors 57

2.1.7 Contact Sensors 57

2.1.8 Thermal Sensors 58

2.1.9 Electrostatic Sensors 60

2.1.10 Summary 60

2.2 Cloth Pickers.....62

2.2.1 Introduction.....62

2.2.2 Walton Pickers.....63

2.2.3 Clupickers.....65

2.2.4 Polytex Pickers.....68

2.2.5 MARS Pick-up Device.....70

2.2.6 Vacuum Pickers.....71

2.2.7 Adhesive Pickers.....73

2.2.8 Other Pickers.....78

2.3 End-Effectors for Flexible Materials.....79

2.3.1 Introduction.....79

2.3.2 $(TC)^2$'s end-effector.....79

2.3.3 Proposed Flexible end-effector.....82

3 Automated Assembly of Men's Dress Pants.....84

3.1 Introduction.....84

3.2 Comparison of Automated and Manual Pants Assembly....86

3.3 Design for Automated Assembly.....87

3.4 Layout of an Automated Factory.....88

3.5 Discussion of Pants Assembly Line.....96

4 Non-linear Cloth Beam Bending Model.....102

4.1 Introduction.....102

4.2 Bernoulli-Euler Beam Model.....103

4.3 Computer Solution of Elastica Problem.....108

4.4 Solution by Shooting Method.....109

4.5 Hints on Solving Heavy Elastica Problem.....114

4.6 Results of Heavy Elastica Program.....119

4.7 Experimental Results of Cloth Heavy Elasticas.....128

4.8 Conclusions.....137

5 Rule for Inter-Picker Spacing 138

5.1 Inter-Picker Spacing Problem.....138

5.2 Inter-Picker Spacing Criterion.....144

5.3 Discussion.....149

6 Expert System for Picker Locations 150

6.1 Introduction.....150

6.2 Picker Placement Expert System.....150

6.3 Rules for Picker Placement Expert System.....155

6.4 Picker Placement Examples.....157

6.5 Adding New Rules to Expert System.....174

6.6 Optimization of Picker Locations.....177

6.7 Hardware Limitations.....178

6.8 Conclusions.....179

7 Discussion of Picker Placement Method.....181

8 Conclusions 184

References.....186

Appendix A: Pants Assembly Instructions.....187

Appendix B: Runge-Kutta Numerical Integration.....216

Appendix C: Computer Code for Elastica Model.....218

Table of Figures

Figure 1-1: The $(TC)^2$ Automated Sleeve Machine.....18

Figure 1-2: A Schematic of The $(TC)^2$ Sleeve Machine.....19

Figure 1-3: $(TC)^2$ Sewing Module of the Sleeve Machine.....20

Figure 1-4: Interlocking Belt Banks.....22

Figure 1-5: $(TC)^2$ Folder Module of the Sleeve Machine.....23

Figure 1-6: Pick and place of a suit sleeve.....30

Figure 1-7: Picker locations for a suit sleeve.....32

Figure 2-1: Thru-Beam Photoelectric Sensor.....39

Figure 2-2: Retro Reflective Photoelectric Sensor.....41

Figure 2-3: An Optical Displacement Sensor.....43

Figure 2-4: Clippard 1022 Proximity Switch.....46

Figure 2-5: Gagne SFX18 One Sided Proximity Detector.....47

Figure 2-6: Gagne SFX8 Thru-gap Adjustable Sensor.....49

Figure 2-7: Gagne SFX8DR Thru-gap Sensor.....50

Figure 2-8: Clippard 1030 Miniature Gap Sensor.....51

Figure 2-9: Gagne Cross Jet Gap Sensor.....52

Figure 2-10: A Capacitive Proximity Switch.....56

Figure 2-11: Contact Sensor on a Conveyor Belt.....59

Figure 2-12: A Walton Cloth Picker.....64

Figure 2-13: Clupicker Made by Jet Sew.....66

Figure 2-14: Polytex Cloth Picker (Swiss made).....69

Figure 2-15: MARS Pick-up Device (Singer).....72

Figure 2-16: Vortex Vacuum Picker (Ann Ito).....74

Figure 2-17: Experimental Adhesive Picker.....76

Figure 2-18: Durkopp Adhesive Picker.....77

Figure 2-19: $(TC)^2$ End Effector for Sleeve Machine.....80

Figure 2-20: Top View of $(TC)^2$ End-Effector.....81

Figure 2-21: Proposed Flexible End-Effector.....83

Figure 3-1: Automated Pants Assembly Line.....90

Figure 3-2: Folding/Sewing Station 3.....95

Figure 3-3: Automated Factory of "Upper Wear."....99

Figure 3-4: Conceptional Ideas of Automated Sewing System...100

Figure 4-1: Diagram of Elastica Beam Geometry.....105

Figure 4-2: Static Equilibrium of Elastica Element.....107

Figure 4-3: Boundary Conditions for BEAM.EXE Program.....111

Figure 4-4: Multiple Solutions for Elastica.....116

Figure 4-5: Elastica 1.5 inch long, μ=0.5.....120

Figure 4-6: Elastica 1.5 inch long, μ=0.8.....121

Figure 4-7: Elastica 1.5 inch long, μ=1.4.....122

Figure 4-8: Elastica 2.0 inch long, μ=0.1.....125

Figure 4-9: Elastica 2.0 inch long, μ=0.3.....126

Figure 4-10: Heavy Elastica Mode Graph.....127

Figure 4-11: Friction Measurement Test.....129

Figure 4-12: Kawabata Bending Stiffness Tester.....132

Figure 4-13: Bending Stiffness of Wool/Poly Suit Material...134

Figure 4-14: Bending Stiffness of Xerox Paper.....135

Figure 4-15: Heavy Elastica Mode Graph and Data.....136

Figure 5-1: Picker Placement for a 6 inch square cloth.....139

Figure 5-2: Excessive Shearing Between Pickers.....141

Figure 5-3: Cloth Catenary between pickers.....142

Figure 5-4: Bistable Cloth Between Pickers.....143

Figure 5-5: Picker Locations for a Pants Pocket.....145

Figure 5-6: Elastica Model of an Area of Pants Pocket.....146

Figure 5-7: Measuring Inter-Picker Deflections.....148

Figure 6-1: Modules of the Expert System.....151

Figure 6-2: Pants Pieces (V) and (P).....158

Figure 6-3: Selecting Edge to be Moved.....159

Figure 6-4: Expert System Highlights the Selected Edge.....160

Figure 6-5: User Inputs Cloth Data.....162

Figure 6-6: First Two Pickers on Pants Piece (V).....164

Figure 6-7: Placing Picker #6 on Pants Piece (V).....165

Figure 6-8: Picker Placement Completed.....167

Figure 6-9: First Five Pickers Placed on the Inseam.....168

Figure 6-10: Picker #6 is Incorrectly Positioned.....170

Figure 6-11: Picker #6 is Correctly Placed on the Inseam....171

Figure 6-12: Inseam Picker Placement is Completed.....172

Figure 6-13: Pants Pocket (S).....173

Figure 6-14: Pickers #2 & #3 on Pants Pocket (S).....175

Figure 6-15: All Four Pickers on Pants Pocket (S).....176

Figure A-1: Manual Assembly of Pants Steps 1 through 7.....191

Figure A-2: Manual Assembly of Pants Steps 8 through 13.....192

Figure A-3: Manual Assembly of Pants Steps 14 through 19....193

Figure A-4: Manual Assembly of Pants Steps 20 through 25....194

Figure A-5: Manual Assembly of Pants Steps 26 through 31....195

Figure A-6: Manual Assembly of Pants Steps 32 through 37....196

Figure A-7: Manual Assembly of Pants Steps 38 through 43....197

Figure A-8: Manual Assembly of Pants Steps 44 through 49....198

Figure A-9: Manual Assembly of Pants Steps 50 through 55....199

Figure A-10: Station 1, Assembling Pocket Pieces.....205

Figure A-11: Station 2, Attaching Pocket to Pants Front.....206

Figure A-12: Station 3, Sewing Front Pocket Edge.....207

Figure A-13: Stations 4 and 5, Attaching Yoke.....208

Figure A-14: Stations 6, 7, and 8, Zipper, Dart, Pocket.....209

Figure A-15: Stations 9 and 10, Pants Inseam & Outseam.....210

Figure A-16: Manual Finishing, Finish Back Pocket.....211

Figure A-17: Manual Finishing, Finish Zipper.....212

Figure A-18: Manual Finishing, Waistband & Crotch Seam.....213

Figure A-19: Manual Finishing, Hem Pant Leg.....214

Figure A-20: Manual Finishing, Attach Belt Loops.....215

1 Automated Apparel Manufacturing and $(TC)^2$

1.1 Introduction

One of the largest consumer manufacturing industries is the apparel industry. In 1982 the U.S. apparel industry had shipments exceeding 15.6 billion dollars. It is an industry that fills an essential need and thus its market is relatively stable. Increasing competition from foreign manufacturers, however, has put many U.S. manufacturers out of business, and put all others on notice.

The apparel industry is typically very labor intensive. About 35% of the cost of apparel in the U.S. is labor[1]. Consequently, the low cost of labor in countries like Taiwan, Korea, Philippines, etc., allows imports to sell for about 20% less than similar U.S. goods even after the application of import duties[2]. Furthermore, the United States' distaste for "protectionism" has left the U.S. apparel industry to fend for itself. In order to compete with foreign imports, U.S. apparel manufacturers have sought to reduce the labor intensity in garment manufacturing through automation.

Although individual companies have attempted some forms of automation, the largest apparel automation program is that of the Tailored Clothing Technology Corp, $(TC)^2$. $(TC)^2$

was founded in 1979 as an organization to advance the state of automation in the apparel industry through research and development. $(TC)^2$ is a consortium of Apparel manufacturers, unions, and the U.S. federal government. Although the list of $(TC)^2$ members has grow to 60+, the original participants were Amalgamated Clothing and Textile Workers Union, Burlington Industries, Collins & Aikman Corp., Milliken & Co., Russell Corp, J.P. Stevens & Co., Greif Companies Div. of Genesco, Hartmarx Corp., Palm Beach Inc., Surgikos of Johnson & Johnson Co., Celanese Fibers Operations, E.E. duPont de Nemours & Co., and U.S. Department of Commerce. The cooperation among these companies, who are normally fierce competitors, is truly commendable. The feeling in the apparel community is that "We are all in the same leaky boat and we must bail and row."

Shortly after the founding of $(TC)^2$, John T. Dunlop of Lamont University and Fred Abernathy of Harvard University conducted an apparel manufacturing study in order to determine the needs of and state of automation in the apparel industry[1]. Dunlop visited many factories and met with some of the leading technologists in the apparel industry. The results of this study are clear. Automation for the apparel industry is available in the areas of

spreading and cutting, but is almost non-existent in joining operations, particularly sewing. The problem with automated joining stems from a lack of ability to "handle" cut pieces of fabric using machinery.

1.2 The Road to Automation

There are a number of steps in apparel manufacturing that precede the joining operation. First, of course, the fabric must be woven and placed on a roll, commonly called a bolt of cloth. Then, the fabric must be "spread" on a spreading table. Spreading is simply stacking a number of lengths of cloth, one on top of the other, so that multiple pieces will be cut during the cutting operation. After the spreading, the cloth is marked and cut. Finally, the cut stacks of pieces are bundled, tied, and sent off to be sewn.

The presence of automation in all operations preceding sewing is very strong. Although it has some problems, automatic spreading is, and has been, readily available for quite some time. Automated marking and computerized cutting have been developed within the past 5 to 10 years. Gerber Co. has a sophisticated cutting machine that uses a thin steel blade. Other companies make laser cutters and water jet cutters. Once the fabric has been cut, however, automation stops and chaos begins. The stacks of pieces are

tied and typically hand carried to an area to be sewn. There the bundles are untied and one seamstress will perform one operation, re-bundle the parts, and send them on to the next operation. Another seamstress will sew her parts and then send them on to the next operation. Transfer lines and conveyor belts are seldom used. The location and/or status of parts at any time is usually not known. The apparel factories have become warehouses of partially finished garments. It can take up to a month in some cases to get the product from the cutting table to the factory door. A need for organization and automation is clear and evident, even in the most modern apparel factories.

Before automation comes into play most factories should get organized. Systems of conveyor belts or overhead trolley racks are commercially available. The use of such systems along with logical factory layout will reduce labor cost, inventory, and speed factory response. Factory operators should put an end to the days of seamstresses trying to work with two bundles of 50 pieces sitting on their table. Automation cannot cure the ills of poor factory planning.

Automation of certain apparel sub-assemblies is quite widely available. Perhaps the most successful of all is the automatic pocket setter. An automatic pocket setter takes a

pre-cut pocket piece; folds the edges under; the operator locates the garment in the machine; the pocket is forced down on to the top of the garment; and an automatic X-Y sewing machine sews the edges of the pocket including backtacks. Automated pocket setters save a great deal of time during pocket setting operations. Another success story in automation is the automated collar machines. The collar machines consists of two separate machines. The first machine automatically fuses the collar to the interfacing. The second machine top stitches the collar edge. Another noteworthy machine is the automated shirt front machine. To make the front of a man's dress shirt, a long straight edge much be folded over, sewn in two places, and then buttons or button holes must be added. The automated shirt front machines can do all of this. Cut pieces of shirt fronts are laid on a conveyor belt with the long straight edges up against a rail. The straight edges are forced through a "horn" which folds the edge. Then a series of one, two, or three sewing machines do the sewing. These machines work remarkably well. Other less shining examples of automation include pillow cases, blanket and towel edging, cuff sewing, and belt making. Although all of these machines, both simple and complex, have met with some success, they still only address a small percentage of all sewing operations. These examples of automation only work

for simple geometry, i.e. straight edges or fixed geometry.

The goal of $(TC)^2$ was to make a quantum leap forward in automated apparel manufacturing. Apparel automation needs more than just task-specific hardware. It needs technology that is capable, at least conceptually, of performing a large range of apparel assembly operations. $(TC)^2$ took to task that issue. To test the viability of such a concept $(TC)^2$ decided to choose one item with which to attempt automation. They chose the sleeve of a man's tailored suit. The choice of a suit sleeve was partly historical (a number of the original sponsors were tailored clothing manufacturers) and partly practical. A suit sleeve has a number of complex operations but it is not a totally three dimensional problem. Making a suit sleeve consists of sewing the inseam including easing for fullness, sewing the vent (cuff), and sewing the outseam. The possibility of actually building a machine, that could automatically do all that, initially seemed very ambitious. An alternative to the sleeve was automating a suit back. But a suit back was rejected on the basis that it was too simple. A man's suit coat sleeve was the goal, and Charles Stark Draper Laboratories was chosen as the organization to build the machine.

1.3 Draper Labs' Automated Sleeve Machine

Seven years and three generations of machines later, Draper Labs had completed their task. The automated sleeve machine has a totally modular design. Separate modules could be combined like boxcars of a train. Three different module types were built: loading, folding, and sewing. A picture of the complete machine is shown in Figure 1-1. The loader module takes one pair of sleeve pieces at a time and lays them on the folder module, see Figure 1-2. The sleeves pairs are stacked like shingles on the loader. The folder module folds the sleeve and positions the edges to be sewn. Once folded, a "door" on the folder module closes on the sleeve and slides it over to the sewing module. The sewing machine, which works in an X-Y fashion, sews the seam contour and eases in fullness where necessary. Once the sewing is done the sleeve is moved on to the next module for additional folding and sewing.

The sewing module is built much like an X-Y plotter, see Figure 1-3. The cloth is captivated by upper and lower belt banks that feed the cloth through the machine in the direction of the transfer line. A sewing machine is mounted on an enlarged frame that moves orthogonally to the direction of the belts. The combination of the sewing machine moving in one direction and the belts moving in the

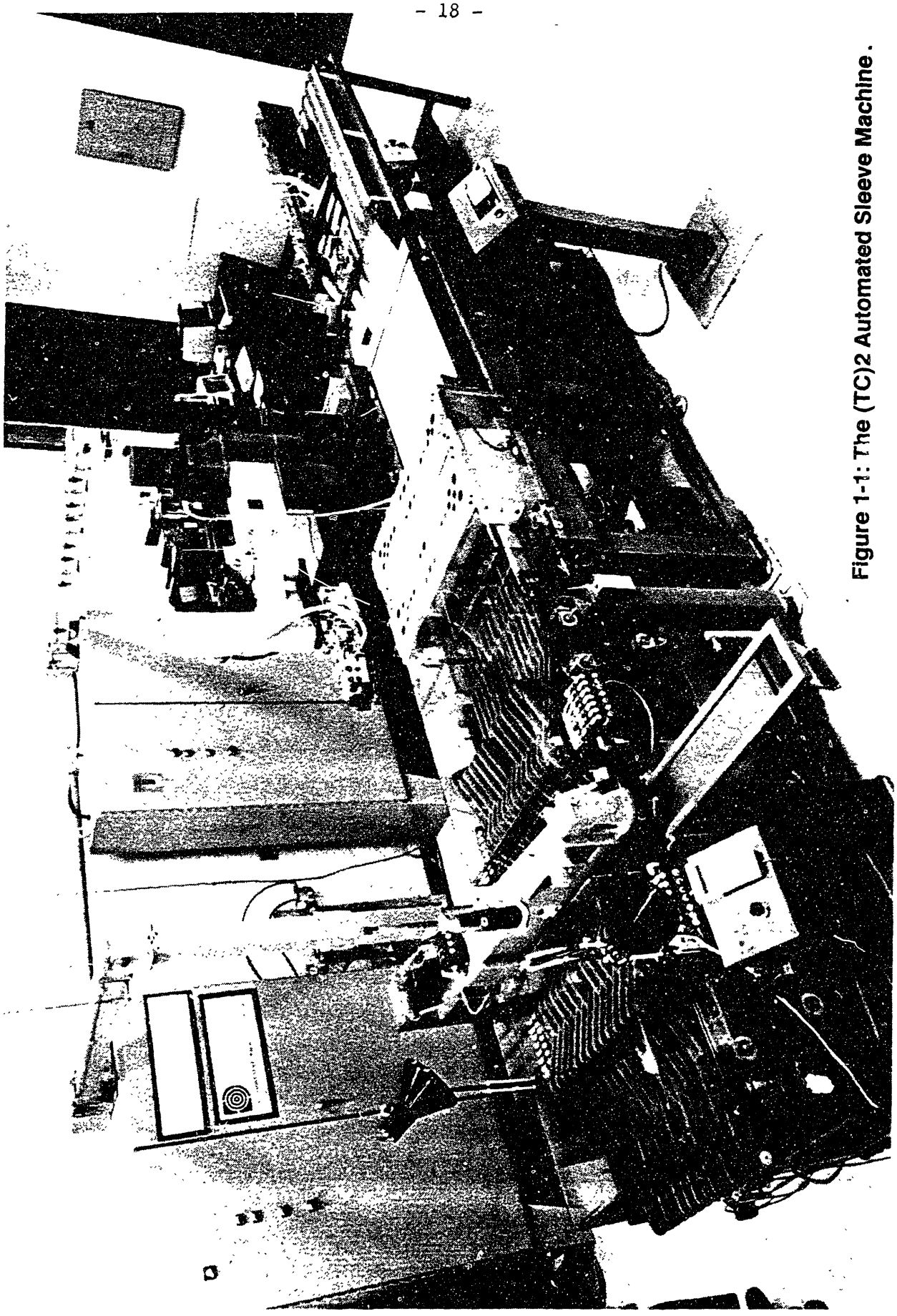


Figure 1-1: The (TC)2 Automated Sleeve Machine.

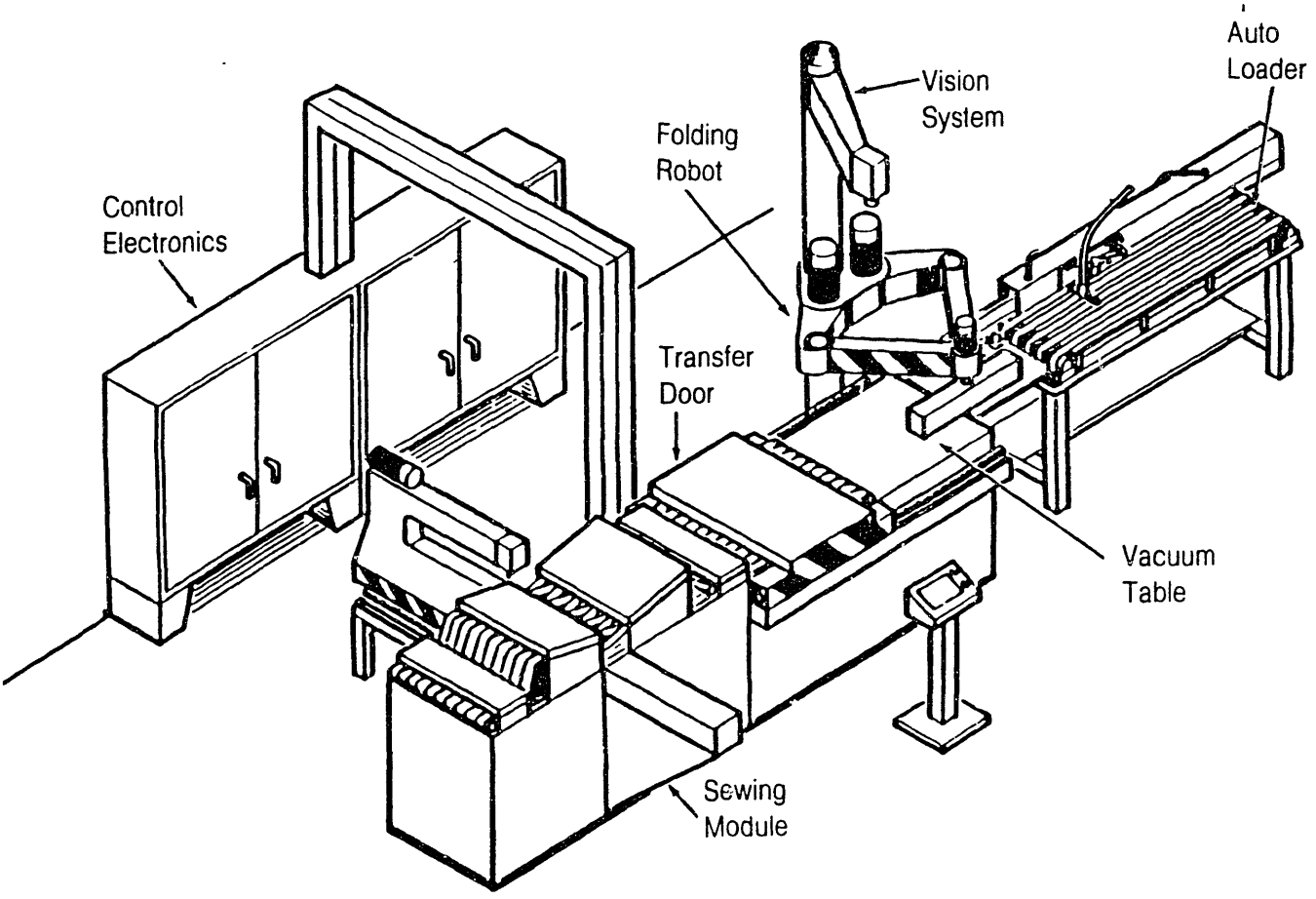


Figure 1-2: A Schematic of The (TC)2 Sleeve Machine .

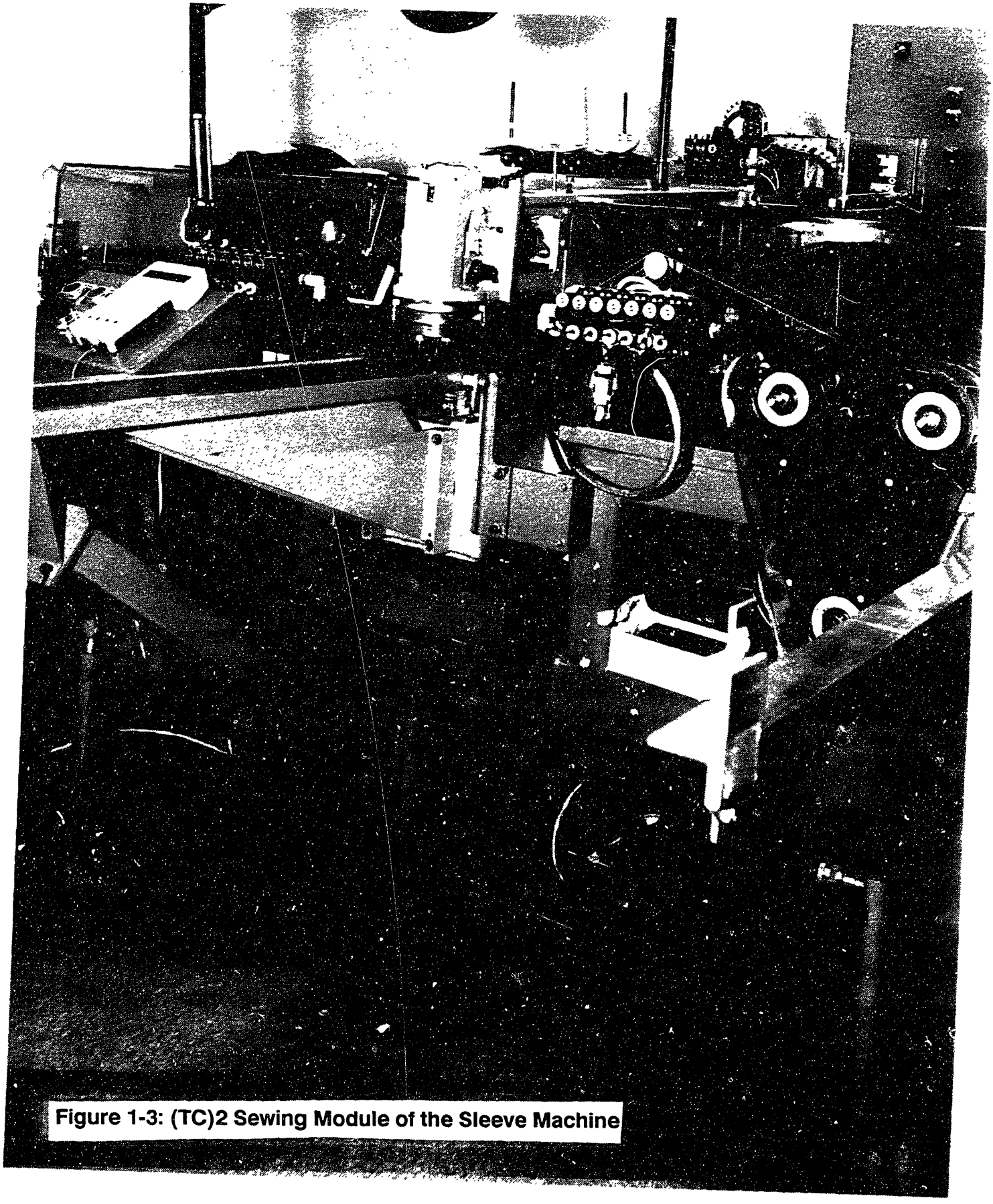


Figure 1-3: (TC)2 Sewing Module of the Sleeve Machine

orthogonal direction allow the machine to sew virtually any two dimensional contour. In order to maintain good local control of the cloth near the needle, the traditional feeddog has been replaced with a "bulldozer" feeddog. The bulldozer has the ability to rotate about the needle so that sewing can proceed in any direction. Since the sewing head must be allowed to traverse the width of the sewing module, the belt banks are divided into two sections, one to the left and one to the right of the sewing head. In between these two belt banks is a "sewing gap" where the sewing head travels. To maintain control of the cloth in the gap a series of interlocking belts were developed. These "interlockers", as they are called, close around the sewing head much like zipper teeth, see Figure 1-4. The combination of interlocking belt banks and the bulldozer feeddog provides precise control of the cloth. The sleeve is eased, when necessary, by slight differential movement between the upper and lower belt banks.

The folding module consists of a robot folder, vision system, vacuum table, and a transport door, see Figure 1-5. To fold and position the sleeve parts with the required precision, it is necessary to have a vision system. The folding table surface is coated with a retro-reflective surface to improve contrast between the cloth parts and the

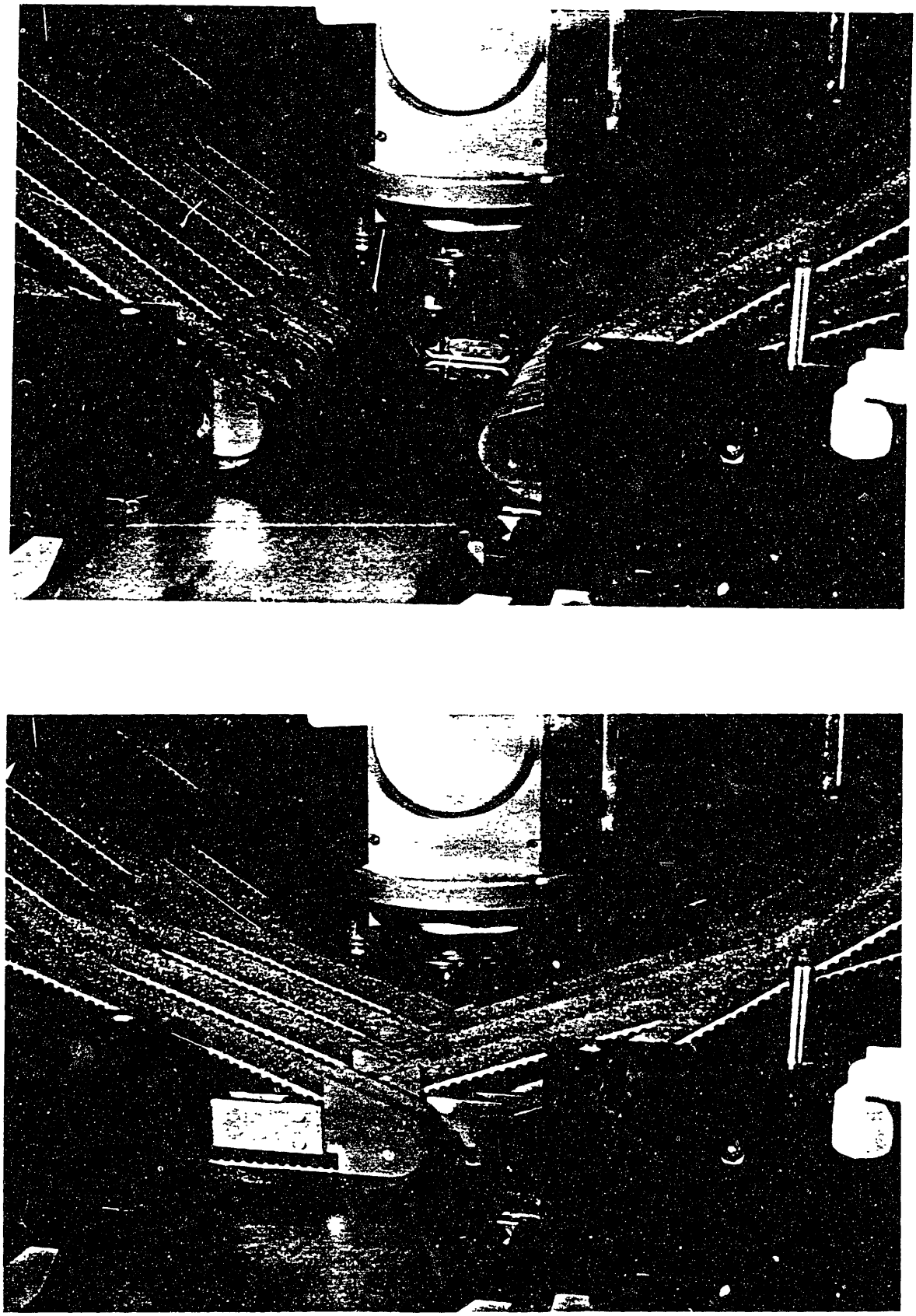


Figure 1-4: Interlocking Belt Banks

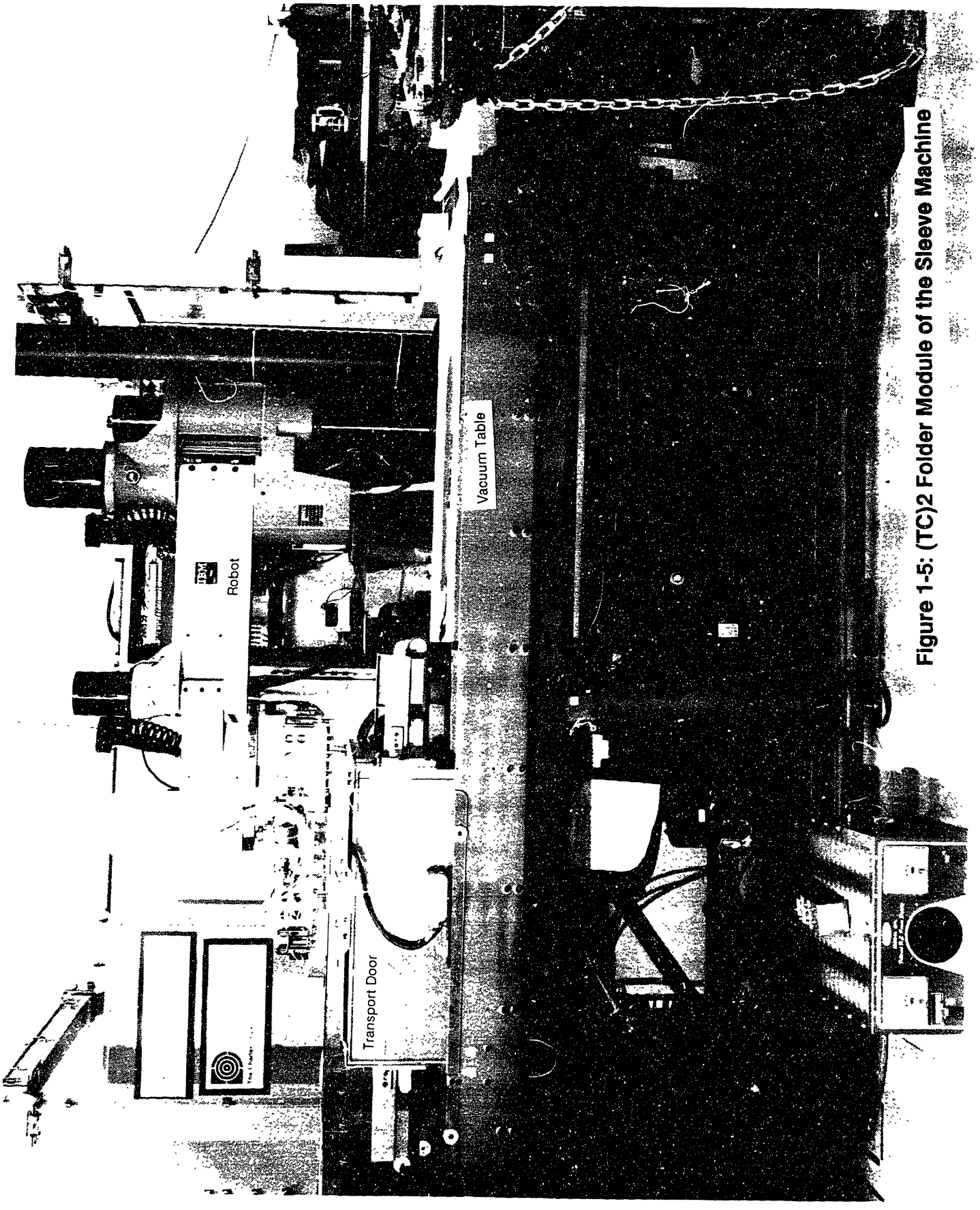


Figure 1-5: (TC)2 Folder Module of the Sleeve Machine

table. The vision system locates the sleeve parts on the table and then tells the robot where key break points are so that the robot can fold the cloth properly. Beneath the folding table is a large vacuum fan that pulls air through the perforated table surface. The vacuum is used to hold the cloth down to the table while the robot is folding. The flow of air through the table is controlled by vacuum gates. The table is divided into 14 rectangular sections and the vacuum in these sections can be turned on and off via computer control. Adjustment of the vacuum pattern is needed when one part of a piece must remain fixed and another part must be moved. At end of a folding sequence the transport door moves over above the folding area, drops down onto the table (sandwiching the cloth against the table), and then slides the cloth over to the sewing module.

Although there are difficulties with both the sewing and folding modules, the folding module in particular needs a great deal of improvement. The sewing module is capable of sewing any contour up to and including circles. Furthermore, it is very easy to program the sewing machine. The robot/vision computers decide where to sew, create a "stitch file" and send it to the sewing computer. This makes the sewing module very flexible. Programming the folding module, on the other hand, is substantially more challenging.

Programming the folder station requires consideration of a number of factors. The first step in automated folding is to take the process that the seamstress uses to make the object and change it so that it is compatible with the folding station. In some ways this is the most difficult step. The automated process usually does not look anything like the original process. Once the method and sequence of folding has been decided then trial "manual" folding begins. Manual folding consists of placing the piece of fabric on the folding table and performing, by hand, the motions of the robot. During manual folding it is decided which vacuum gates to have open.

Once the manual folding is complete, the next step is to "teach" the robot the same sequence. The manipulator is moved to the point where it is to pick up the cloth. The pickers are activated and the robot picks up the edge of the cloth and moves it to the target location. Three things could go wrong in this process: 1) The main section of the cloth did not remain stationary during the folding; 2) The cloth did not lie flat after the fold was completed; or 3) The vacuum did not hold the cloth in place when the fold was completed. These problems must be solved by changing the pick up location and folding trajectory. Once the robot

folding sequence is "ironed out" then the appropriate vision program must be developed so that the robot can compensate for the parts not showing up at the same location on the table every time. A detailed discussion of vision systems and vision algorithms is outside the scope of this thesis.

The difficulty of developing complete folding sequences has led to a serious lack of flexibility in the folding module. This lack of flexibility is not due to the lack of capability of the hardware, it is due to amount of time and effort that must be expended to reprogram the folder for a new size, style, or product. The lack of understanding of the folding process is the cause of most of these difficulties. The amount of trial-and-error involved in determining folding sequences must be reduced. It is important to realize that the process of developing folding sequences, i.e. vacuum patterns, picker locations, manipulator trajectories, etc., is not a black art. There is a reason to turn the vacuum on at a certain location at a certain time. There is a reason to place pickers at certain spots and not at others. The mechanics behind each folding sequence depends on the cloth properties, part geometry, and system environmental factors. The important properties of cloth for folding are believed to be air permeability, bending stiffness, tensile stiffness, shear stiffness,

creasability, coefficient of friction, and weight. These cloth properties are not usually well defined. Even if the properties were all easily available, however, ignorance of the behavior of the cloth during folding leads most people to believe that folding is a black art. This position is further reinforced by the great range of values that cloth properties can have for different types of cloth. Therefore, the key to shortening the development time of folding sequences is to develop a basic understanding of the physics and mechanics of cloth folding.

1.4 Expert System for Folding

One possible solution to the problem of developing folding sequences is an expert system for cloth folding. Expert systems, initially developed for the medical community, have become increasingly common in other fields.

An expert system is just a computer program that uses radically new programming logic. Conceptually, the expert system would use all existing knowledge about folding (written as a set of rules), and use this knowledge to develop new folding sequences. The ideal folding expert system could take high level information like part geometry and the edges to be aligned, and develop complete vision, robot folding, and sewing sequences. Even though such a

ideal system may never be realized, less-than-ideal expert systems would also be very useful. For example an expert system that could determine when to turn vacuum gates on and off. Alternatively, an expert system could determine where to place pickers on a cloth piece. An expert system could determine how high to lift the cloth off the table during folding so that the cloth folds instead of bunching. Any amount of help in determining folding sequences is valuable.

The foundation of any expert system is its *rules*. The rules provide the expert system the basis for making decisions. For example, consider the task of determining which vacuum gates to open. A typical rule might be, "Turn on vacuum gate if it has cloth over it." But the vacuum must be turned off along the edge that has to be move. Therefore, a second rule might be, "Don't turn vacuum on if the edge to be moved lies above that gate." Obviously the two rules will be in conflict under some conditions. Most expert systems allow the programmer to assign a confidence level to each rule. So rule number one might have a 70% confidence level and rule number two might have a 90% confidence level. When the two rules conflict, the second rule would supersede. Of course real expert systems have a great number of rules and are more complex than just described.

Rules for expert systems are derived from two sources. The first source is the *expert*. An expert is simply a person that has a high degree of knowledge in a certain "domain." An expert gains this knowledge through the course of his work and studies. Although it is not an easy task, this knowledge can usually be written down as a set of rules that are similar to the rules that the expert uses heuristically. The other source of rules is derivation from *scientific principles*. For example, in the above example of setting vacuum pattern it is known that the pressure difference across the cloth is proportional to the air flow rate. A rule based on this is a rule based on science. The use of scientific rules is obviously very desirable. An expert system is only as good as the rules it uses. Searching for the most accurate and reliable rules of folding will help to develop the most accurate folding expert system possible.

1.5 Expert System for Picker Locations

One major problem of folding is the determination of picker locations. As an example of a simple robot movement, consider aligning two edges of an inseam of a sleeve. Figure 1-6a shows the two halves of a typical suit sleeve. Before the sleeve can be sewn, the left edge of piece No. 2

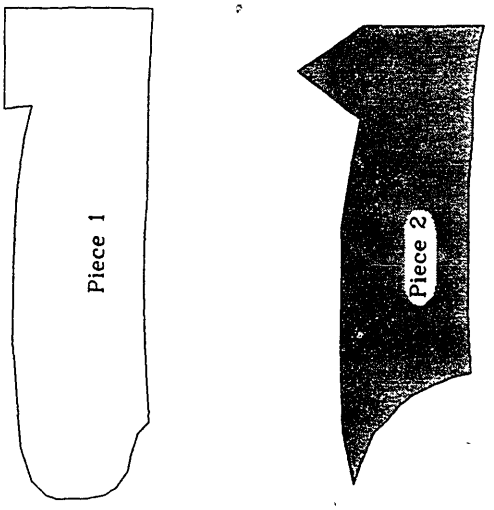


Figure 1-6a: Two sleeve pieces prior to alignment.

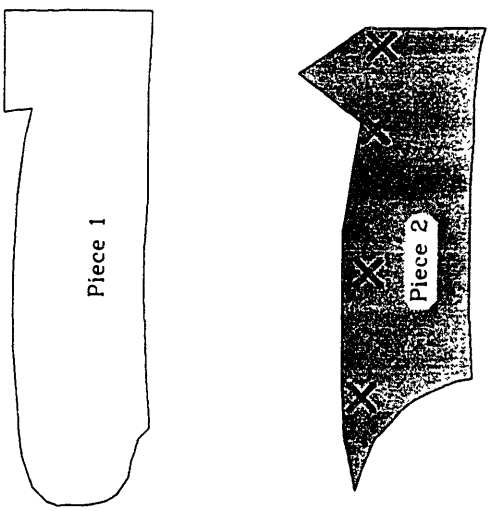


Figure 1-6b: Picker locations marked by X's on piece number 2.

must be aligned with the left edge of piece No. 1. If this were done on the $(TC)^2$ machine, the vacuum would be turned on under piece No. 1 so that it remains stationary while piece No. 2 is moved. Piece No. 2 is moved by picking up along the left hand edge and dragging the piece to the left. The right half of piece No. 2 would remain in contact with the table surface and be dragged along. Once piece No. 2 is positioned above piece No. 1 it is then lowered, supposedly aligning the two left edges. Unfortunately, if the pickers are placed as shown in Figure 1-6b, the sleeve will probably end up as depicted in Figure 1-7a. As shown, a poor choice of picker locations has caused the sleeve vent to be folded under. A better choice of picker locations is shown in Figure 1-7b.

In the above example, the words "poor choice" and "better choice" were used in regard to the picker locations. One might rightly ask what is meant by a poor choice or a better choice. A **good choice of picker locations** is one where the cloth can be picked up, moved to a new location, and put down without having any part of the cloth get folded under or seriously distorted. There is no **best** choice of picker locations. However, there are picker locations that will not work, e.g. the one in Figure 1-6b. In one sense, the use of an infinite number of pickers along the edge to

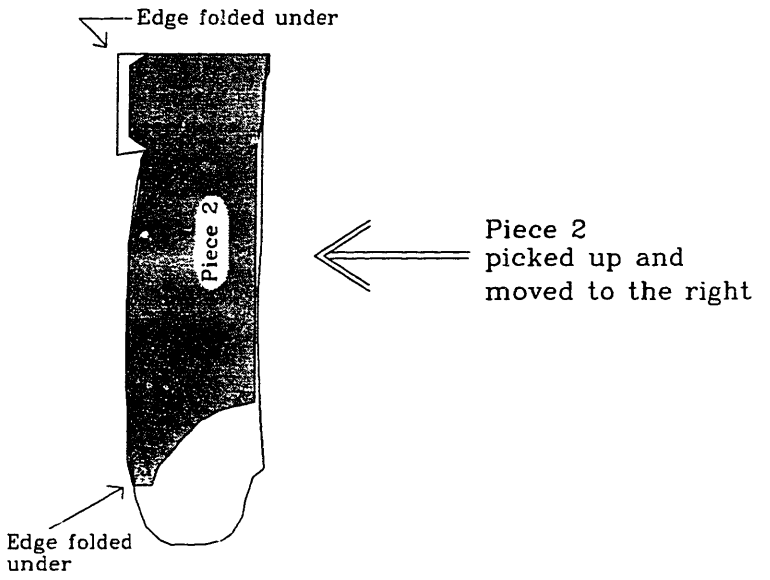


Figure 1-7a: Two sleeve pieces after failed alignment. Note the edges that were folded under on piece 2.

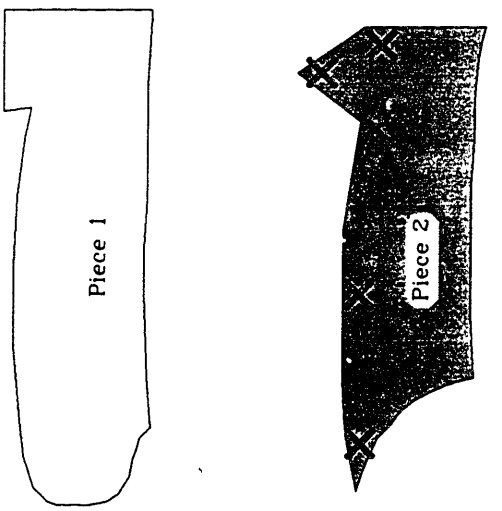


Figure 1-7b: Better picker locations for piece 2. The pickers are supporting the areas that were folded under in the previous figure.

be moved is optimum. With an infinite number of pickers, the cloth cannot get folded under. If cost is considered, an infinite number of pickers ($20-$400 each) is certainly not optimum. Even when using a finite number of pickers, more is not necessarily better. It is true that more pickers give better cloth control; but the better cloth control is not needed if adequate cloth control can be achieved with fewer pickers. It is important to realize that the cost of pickers includes cost per picker, cost of the mechanism to move the picker on the end-effector, cost of support equipment (pneumatic and/or electrical controls), and the cost of the robot (the size of which is related to the weight of the end-effector which is in turn related to the number of pickers). It is best to use a few pickers at locations that are chosen prudently.

The choice of good picker locations is not particularly obvious. Current $(TC)^2$ method for determining picker locations is by trial-and-error. Consider again the picker locations shown in Figure 1-7b. This represents a good choice of picker locations for suit material. If the sleeve was made of a stiff material, like leather, this would be a poor choice of picker locations. With a stiff material, fewer pickers could be used. On the other hand if the sleeve was to be made out of a limp material, like silk, more pickers would be needed for adequate cloth control. So

the choice of picker locations depends on both geometry and cloth properties. It is very difficult for a person (even an experienced one) to determine picker locations for a complex part of various cloth properties. That is the reason for going to an expert system.

The expert system considers all the rules at all time; but the trick to developing a good expert system is to find the rules in the first place. One way to find the rules is to consider simple problems that approximate local behavior of the cloth. In Figure 1-7b for example, what the right half of piece No. 2 is going to do is not worth considering because it has relatively little effect on the behavior of the cloth near the pickers. On the other hand, the coefficient of friction of cloth against the table surface is very important because it affects the behavior of the cloth edge when it is put down. By considering models of the cloth that approximate the cloth behavior near the pickers, rules for an expert system can be determined that will be capable of selecting suitable picker locations.

It might be argued by some people that an approximation of the cloth pick up process is not necessary because an "exact" solution can be obtained using a non-linear finite

element program. There are a few problems with this argument. First, this author knows of no non-linear finite element program capable of handling the problem (because it involves frictional contact and very large deflections). If such a program did exist, it would require an unreasonably long execution time. Furthermore, even if such a program did exist, it would not tell us where to place pickers. Such a program would simply take cloth geometry and picker locations and display the deflections. If the picker locations were good, then there is no problem. But if the picker locations were bad, there is no suggestion of where the picker should be moved to. With the use of approximate models near the pickers, the picker locations can be determined through applications of the rules. The advantages of using rules based on approximate model will be demonstrated later.

1.6 Overview

The goal of this thesis is to develop a methodology for determining picker locations. Chapter 2 presents an overview on automated apparel manufacturing hardware, including sensors, pickers, and robot end-effectors. Chapter 3 demonstrates the potential application of $(TC)^2$ technology to pants manufacturing. The current method for manual pants assembly will be presented, then a proposed

method for automated assembly using the $(TC)^2$ machines. Chapter 4 presents an analytical model that is used to develop an important rule for how close the pickers need to be placed to the edge of the cloth during cloth pickup. Chapter 5 presents a heuristic model that is used to determine a rule for the distance between pickers. Chapter 6 integrates the results of the last two chapters, and demonstrates the use of the rules developed in chapters 4 and 5 for picking up some of the pants pieces in chapter 3.

2 Automated Apparel Manufacturing - Basic Hardware

There are three basic elements in an automated apparel folding system: the sensors, the pickers, and the end-effectors. The sensors are needed to locate the cloth. They can be anything from single point sensor up to two or even three dimensional vision systems. The pickers are necessary for single point pick up. A number of pickers placed on the end of a robot is an end-effector for cloth. These three components put together make up the essential hardware used for most automated apparel folding. A discussion of each of these three components follows.

2.1 Sensors for Flexible Materials

2.1.1 Introduction

The increase in automated manufacturing of textiles and other flexible materials has developed the need for a special class of sensors. These sensors must be able to detect the edge of a material that is very limp, air permeable, light permeable, very thin, acoustically soft, and has "fuzzy" edges (woven materials). Contact sensing is usually not reliable. Even a couple of grams of force is sufficient to deflect the edge of most flexible materials.

Traditional non-contact methods of sensing are capable of sensing flexible materials, but can have some difficulty due to the nature of these materials.

There are several types of non-contact sensors that are useful for detecting flexible materials. These methods are optical (single element and vision systems), pneumatic, and ultrasonic. These methods and the available sensors are discussed. Also other methods of sensing are mentioned and an explanation is given as to why they will not work very well on flexible materials.

2.1.2 Photoelectric Sensors

Photoelectric sensors are by far the most widely used sensors for flexible materials. They are fast, fairly inexpensive, and accurate. A typical application is edge detection of a single sheet or ply of material. For example, a mechanism might advance a piece of cloth until the cloth was detected by a photosensor, then the edge of the cloth would be hemmed. There are two main categories of photoelectric sensors, thru-beam and reflective.

A thru-beam sensor consists of a light source and a photodetector set up in a "pitch-catch" arrangement, see Figure 2-1. When an opaque object blocks all or part of the

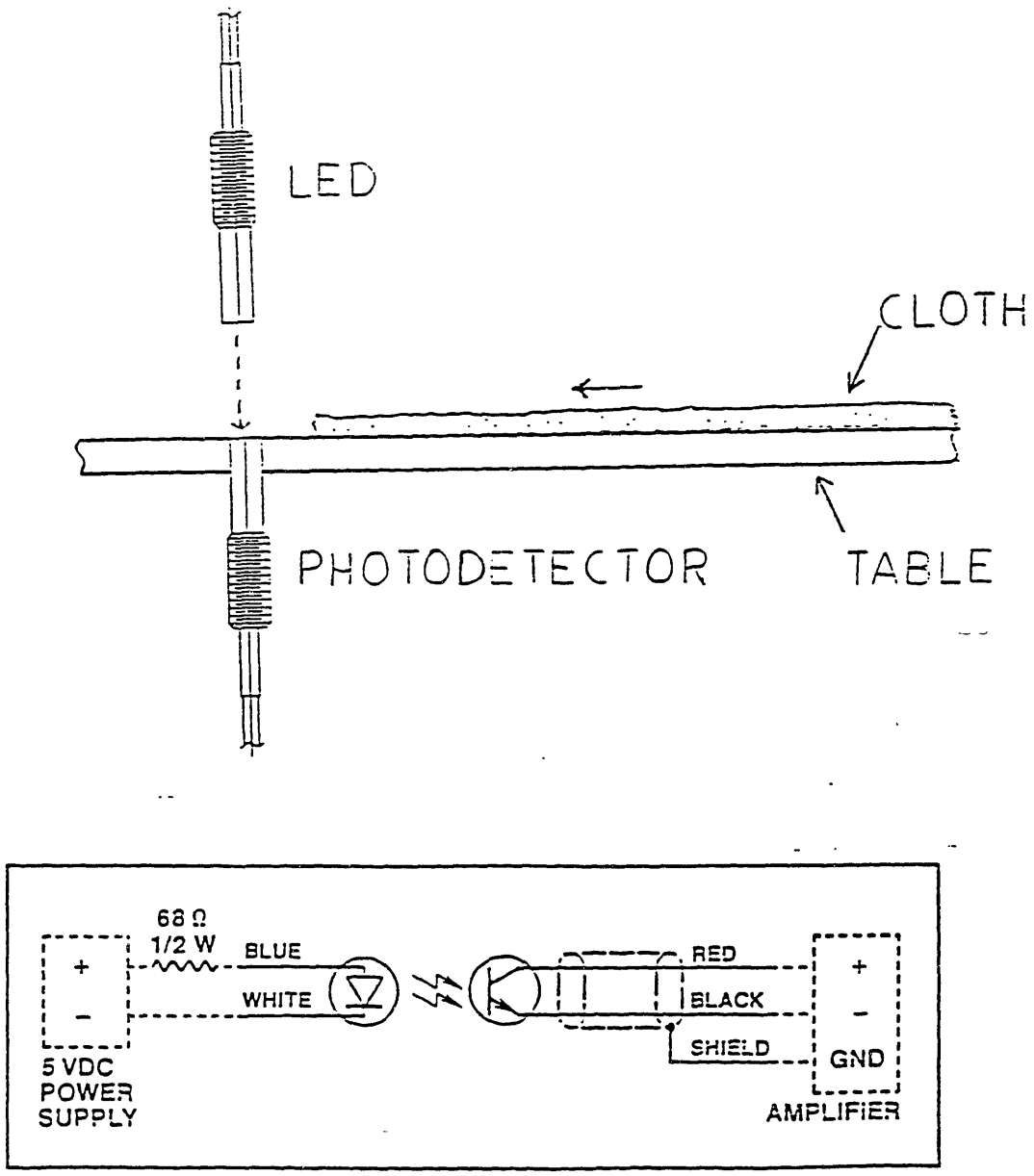


Figure 2-1: Thru-Beam Photoelectric Sensor being used for edge detection.

light from reaching the photodetector, the sensor is triggered. The amount of light that triggers the sensor is adjustable on some models. Flexible materials are not totally opaque, but optical sensors can still be used although some adjustment may be necessary for different materials. Small fiber optic thru-beam sensors may have sufficient accuracy that such adjustment is not necessary.

There are certain advantages and disadvantages to thru-beam sensors. One advantage is that they can travel longer distances than reflective sensors because the light beam only travels half the distance. Also, no light is lost by reflecting off of a surface. Disadvantages are that mounting and alignment are critical. In addition, it is sometimes difficult to incorporate thru-beam sensors into a system because the light source must be mounted on one side of the object and the photodetector on the other side.

The most reliable form of reflective photosensor is the retro-reflective scan, see Figure 2-2. The retro-reflective photoelectric sensor uses a retro-reflective target as the object to be detected. A retro-reflective material is a material that reflects light back along the same path as it came, like a road sign or bicycle reflector. In this way, a larger percentage of the light reaches the photodetector.

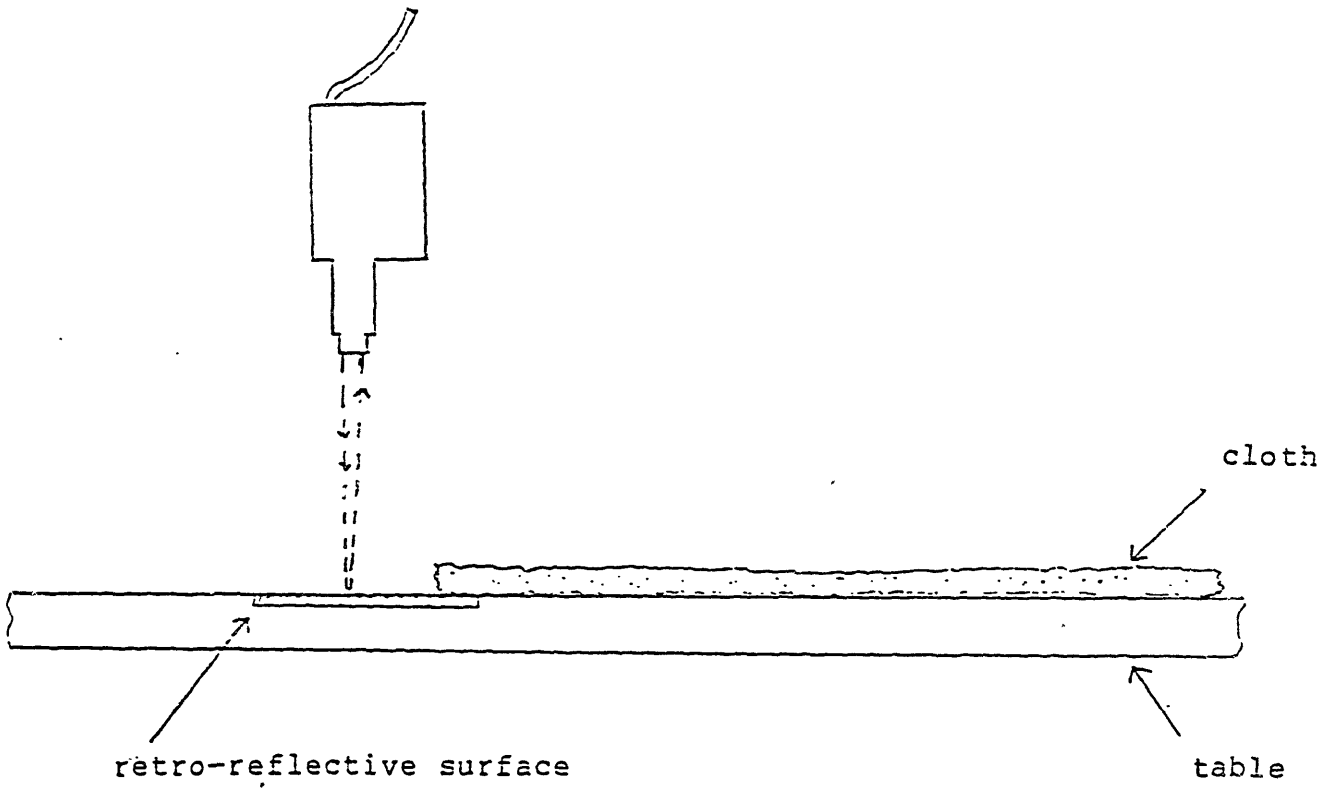
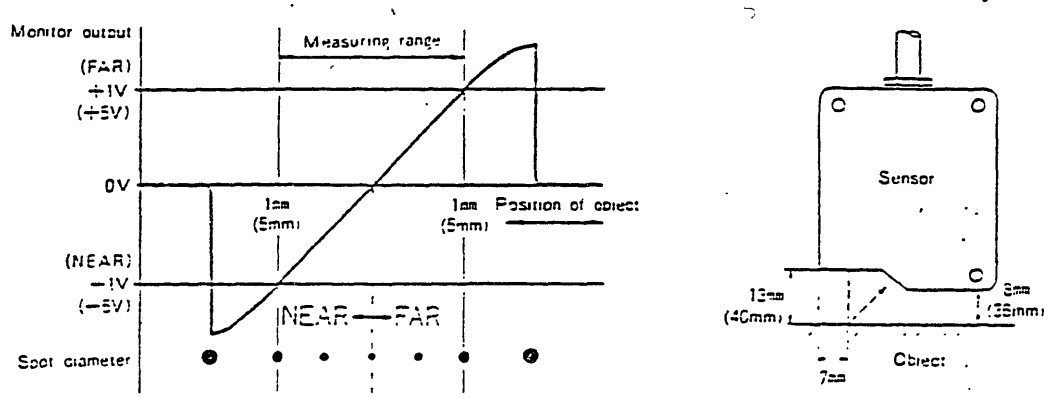


Figure 2-2: Retro Reflective Photoelectric Sensor.

This design allows the light source and the photodetector to be packaged side by side. Operation is quite simple. When no object is present, light emitted from the light source is reflected off the target into the photodetector. When an object is present, very little light reaches the photodetector and the sensor output is switched.

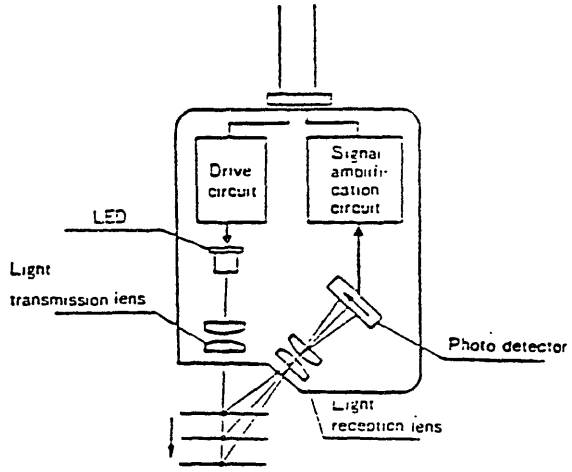
The biggest advantage of retro-reflective photoelectric sensors is that sensing equipment need only be mounted on one side of the object. This advantage makes machine design simpler and less expensive. Furthermore, it makes position adjustment of the sensor very easy, since alignment and angle are not critical. For these reasons, retro-reflective photoelectric sensors are the most popular sensors among manufacturers of automated textile equipment.

There is one special type of photoelectric sensor for distance measurement, it is called an optical displacement sensor, see Figure 2-3. The sensor uses an LED, a photodetector, and the method of triangulation to measure the position of the sensed object. Keyence Corp. makes two such sensors. One of their sensors has a range of ± 1 mm (0.040 inch) with an accuracy of ± 0.005 mm (0.0002 inch). The other sensor has a range of ± 5 mm (0.2 inch) with an accuracy of ± 0.020 mm (0.0008 inch). Used in pairs, these



Bracketted vaive for PA-1801U

Principle diagram



Operation principle

Light emitting diode (LED) beam, narrowed by lens, is applied to object.
Diffused reflection is focused through reception lens; spot image is formed on photo detector.
Spot shifts with object displacement.
Spot position on photo detector is converted to electric signal, which is transmitted to controller.
Controller outputs displacement value unaffected by object reflection factor, owing to its computation circuit.

Figure 2-3: An Optical Displacement Sensor.
(diagram from Keyence Corp. sales literature)

sensors can be used to determine material thickness. Another possible use would be non-contact surface texture profiling. These sensors have very limited range, therefore many applications would be better served by an ultrasonic sensor. The cost of these sensors is high, therefore other alternatives should be investigated before their use is seriously considered.

The most sophisticated form of a photoelectric sensor is a vision system. A vision system can be used to take data over a large area. To be useful, the vision data must be processed by a computer to extract the desired information. This is usually complicated and expensive. However, it is sometimes the only way to get the needed information about an object. This is particular true of objects that are not a simple shape, like rectangles or polygons. A discussion of vision systems is beyond the scope of this paper.

2.1.3 Pneumatic Sensors

All pneumatic sensors work on the principle that any liquid or solid object will present some resistance to the flow of an air stream. This resistance will divert the air

stream and cause a suitable pressure rise or drop somewhere in the sensor. This change in pressure indicates the presence or absence of the object being sensed.

There are two main types of pneumatic proximity sensors, the single sided back-pressure type and the two sided thru-gap type. These types are similar to the reflective and thru-beam photoelectric sensors. Pneumatic sensors can become very complex when the designer is trying to achieve certain goals such as fast action, long range, etc. However, the basic design is always one of the two main types. Therefore, the discussion will be limited to the basic characteristics of these two main types.

Single sided back-pressure proximity sensors only work over a very short range. Figures 2-4 and 2-5 show two different types of these sensors, made by Clippard and Gagne Associates, respectively. Both manufacturers rate their units to operate at a distance of 0.1 inch (2.54 cm) from the sensed object. The output pressure, at this distance, is 7.5 and 4 inches (19 and 10 cm) of water for the Clippard and Gagne, respectively. This pressure is less than 1/4 psi. Proper sensing requires either a fluid amplifier or a very sensitive pressure detector. Both these companies sell such devices. The companies claim the units work reliably even at such low pressures.

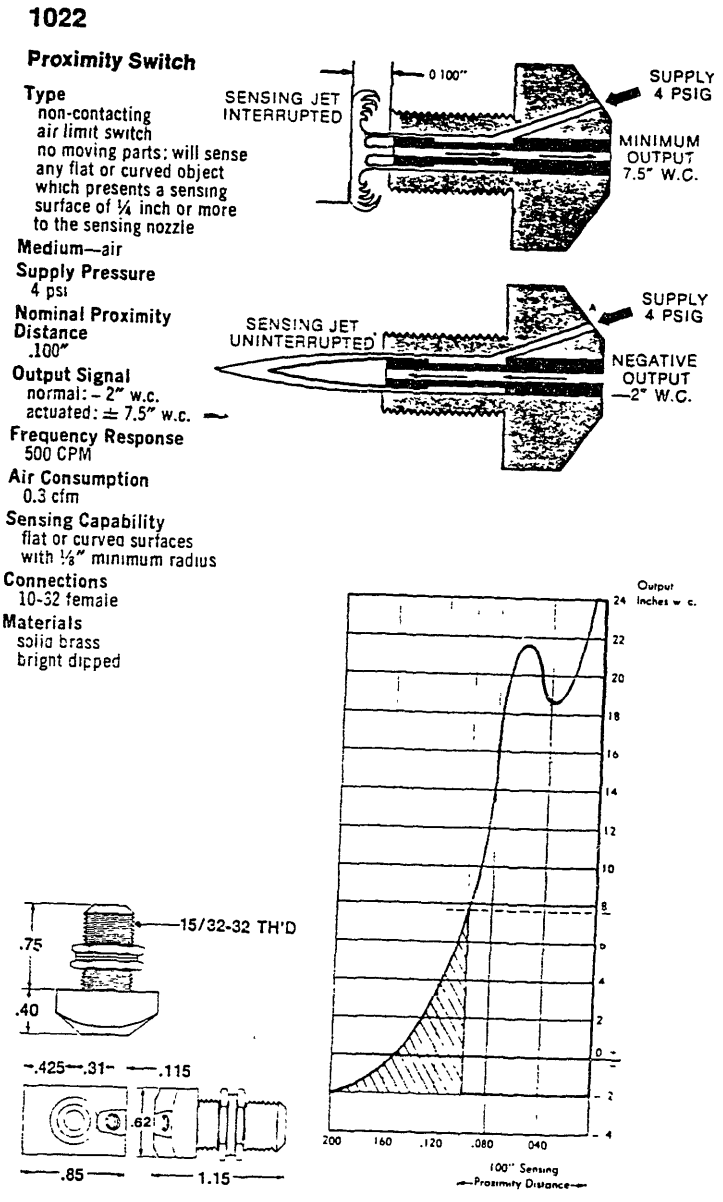
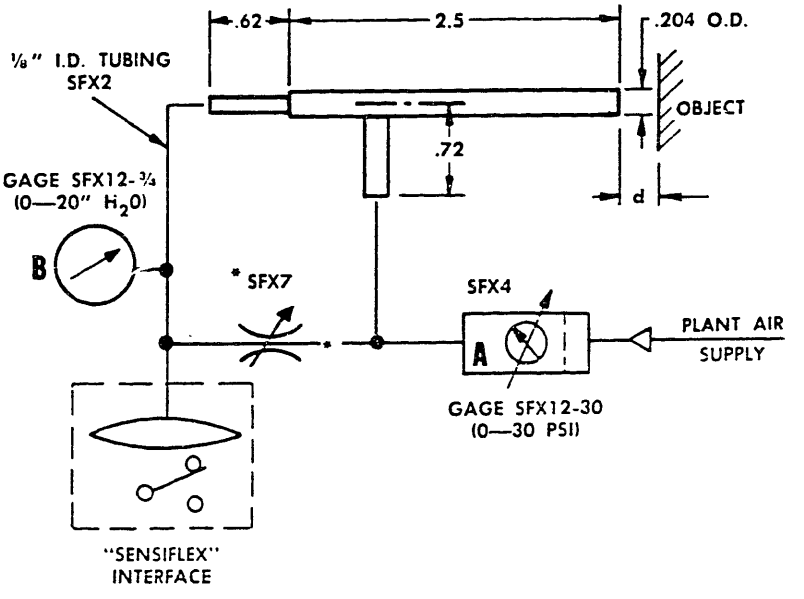
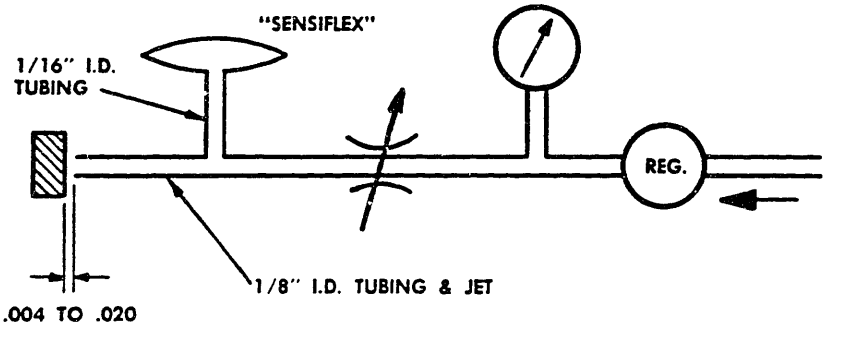


Figure 2-4: Clippard 1022 Pneumatic Proximity Switch
(Diagram from Clippard Inc. sales literature)



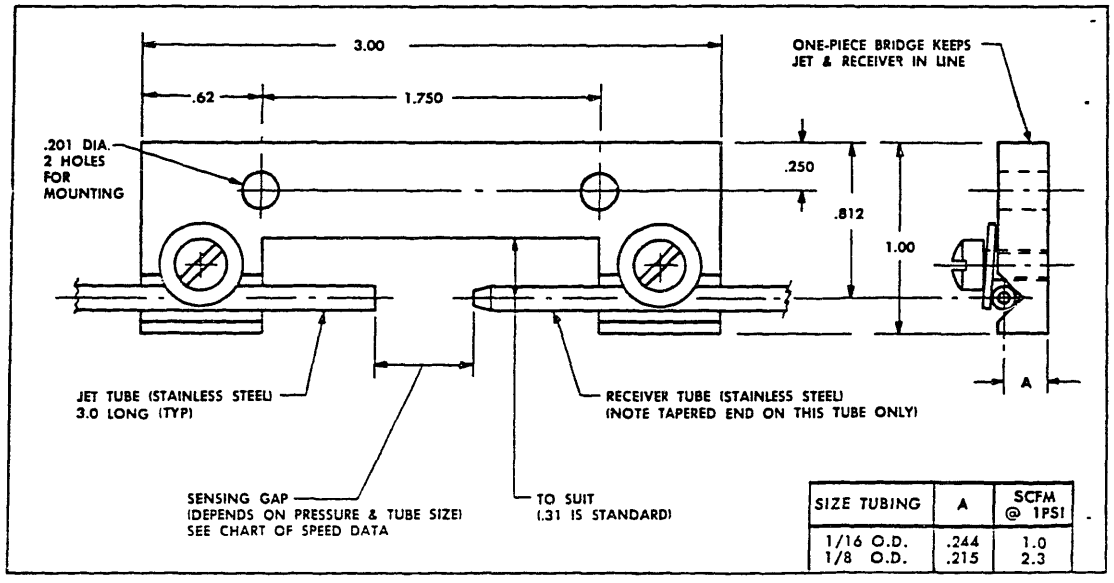


"Probe" Method—from One Side of Object

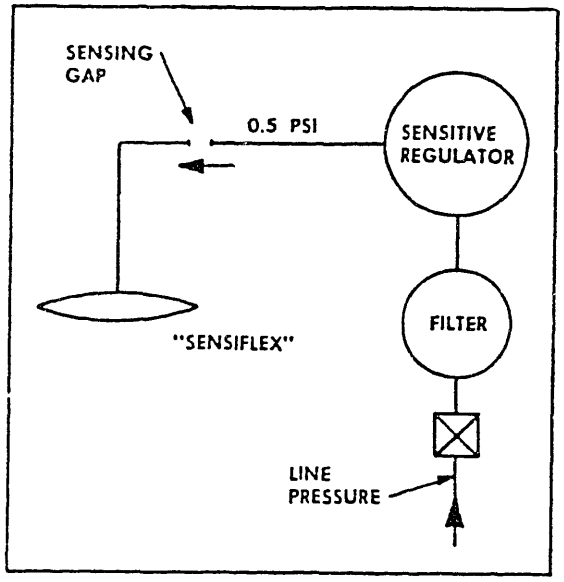
Figure 2-5: Gagne SFX18 One Sided Proximity Detector.
(Diagram from Gagne Associates Inc. sales literature)

The pneumatic gap sensors are typically more sensitive and can operate over longer distances than the back-pressure proximity sensors. Figures 2-6, 2-7, and 2-8 show three different commercial gap sensors. The gap in the sensor in Figure 2-6 is adjustable from 0 to 1.5 inch (3.81 cm) showing that these sensors do have a much larger range than the back-pressure style. The Clippard sensor, Figure 2-8, shows that for a 1/4 inch (.64 cm) gap the output pressure is about 25 inches (63 cm) of water (4 psi input). This is much more sensitive than the back-pressure sensor which had an output of only 7.5 inch (19 cm) of water at 0.1 inch (.25 cm). These sensors have an obvious alignment problem across the gap. Also, machine design is more difficult because sensing equipment must be placed on both sides of the object being sensed.

Gagne Associates make one very long range Cross Jet sensor, see Figure 2-9. In this sensor an air jet is used to deflect the path of air stream that is part of a gap sensor. When the path of the air jet is blocked by an object, the air stream flows through the gap sensor. When the air jet is not blocked, the air stream is deflected and there is no output from the gap sensor. This form of sensor is claimed to be useful for up to 18 inches (46 cm) with an air jet, or 36 inches (91 cm) using a blower.



SIZE TUBING	A	SCFM @ 1PSI
1/16 O.D.	.244	1.0
1/8 O.D.	.215	2.3



BASIC PLUMBING
(gives rapid action or long gap)

Figure 2-6: Gagne SFX8 Thru-Gap Adjustable Sensor.
(Diagram from Gagne Associates Inc. sales literature)

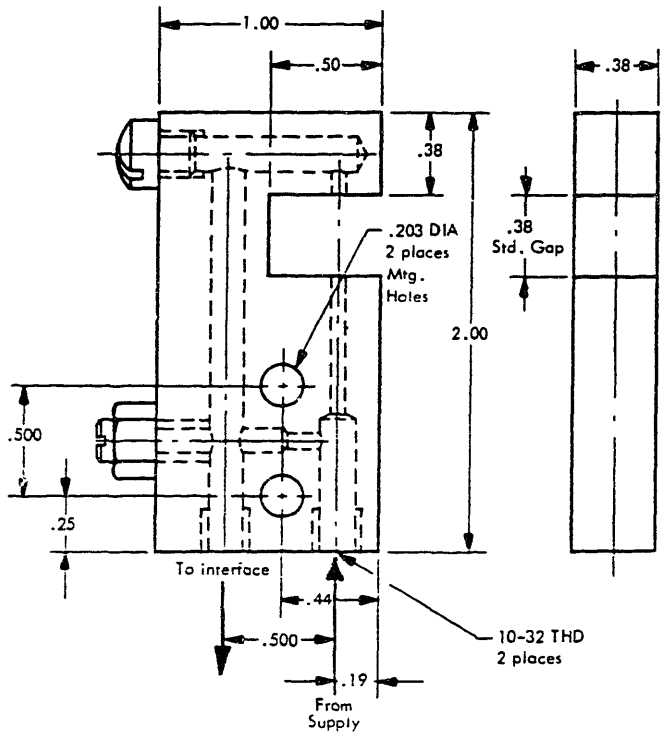


Figure 2-7: Gagne SFX8DR Thru-Gap Sensor.
(Diagram from Gagne Associates Inc. sales literature)

1030

Miniature Gap Sensor

Type
non-contact
positive pressure sensor
will sense any flat or round object with a 1/32" minimum radius
produces a positive signal when no object is present and a negative signal when an object interrupts its sensing system

Medium—air

Supply
0.5-5 psi

Output
-3" H_2O to ± 26" H_2O @ 4 psi

Frequency Response
1000 cpm

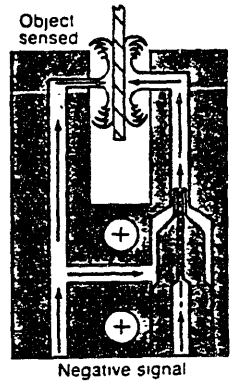
Air Consumption
¼ cfm @ 4 psi

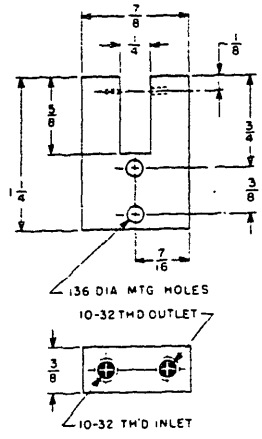
Temperature Range
-30° to 230°F.

Sensing Capability
flat or curved surfaces with 1/32" minimum radius. May be used for up to 4" gap. Ask for 1030 Design Tips.

Connections
10-32 female

Construction
solid brass
bright dipped





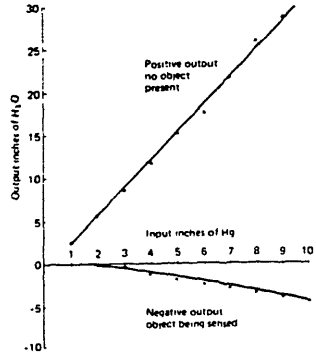


Figure 2-8: Clippard 1030 Minature Gap Sensor.
(Diagram from Clippard Inc. sales literature)

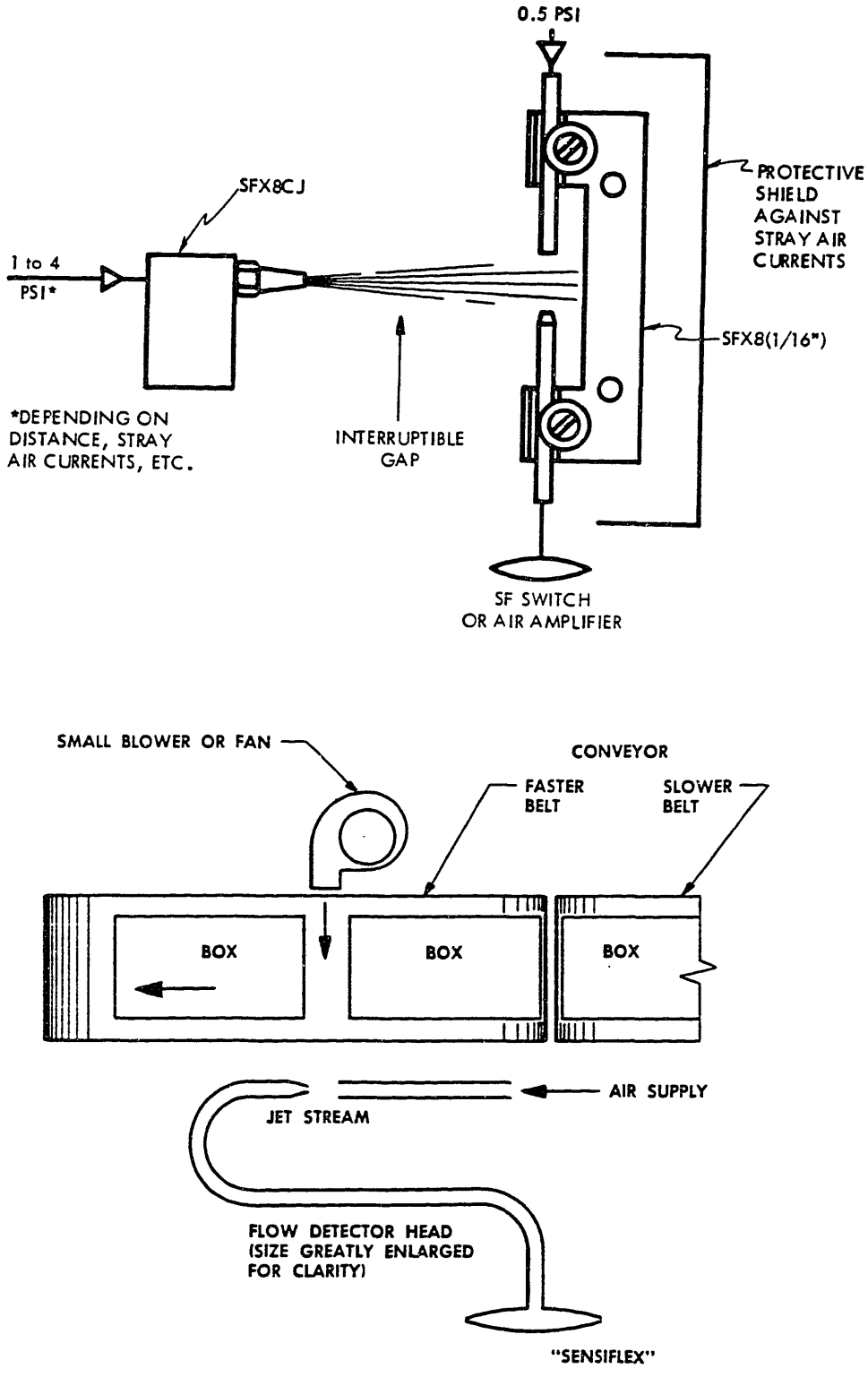


Figure 2-9: Gagne Cross Jet Gap Sensor.
(Diagram from Gagne Associates Inc. sales literature)

The use of pneumatic position sensors for flexible materials can have certain drawbacks. If an air jet, which is part of a sensor, is blowing on a flexible material it may have a tendency to cause the edge of the material to flutter. This would make edge detection difficult. This can be minimized by having the air jet positioned so that it tends to force the flexible material flat against the surface of the machine. Another possible problem is air leakage through permeable textiles. At best this will decrease the sensitivity of the sensor and at worst it will make them unusable (Consider a material like overlay lace). On the good side, pneumatic sensors are cheaper than photoelectric sensors. Also, many actuators in automated equipment are pneumatic and the use of pneumatic sensors allows them to be activated without electronics.

2.1.4 Ultrasonic Sensors

Ultrasonic sensing devices are capable of providing accurate distance measurement for the position of flexible materials. Ultrasonic sensors work by emitting a pulse of ultrasonic waves, having the wave reflect off an object, and catching the reflected waves. Using the elapsed time and the speed of sound, the distance to the object can be

calculated. Typically the same transducer can both emit and detect the sound waves. Ultrasonic frequencies are used so that interference from background noise is minimal.

Perhaps the most well known use of ultrasonics is in the Polaroid SX-70 series cameras. Polaroid uses the distance information from the sensor to focus the camera lens. This sensor has a range of 0.9 to 35 feet (0.3 to 10 m) with and accuracy of ± 0.12 inch (3 mm) up to 10 feet (3 m) (± 1% over the entire range). The accuracy and the lower limit of the range is limited by the electronic circuitry. The author verified (using a test kit) that the Polaroid sensor does work on nearly all types of cloth. However, this sensor has too much range and not enough accuracy for automated assembly of flexible materials.

Other companies manufacture ultrasonic sensors specifically for automation, for example Yodel Technology. They make an ultrasonic transducer with a range of 1 to 10 inches (2.5 to 25 cm) and an accuracy of 0.005 inch (0.1 mm). The thickness of cloth is in the range of 0.015 to 0.030 inch (0.4 to 0.8 mm). Therefore, it would be possible to detect the difference between a single layer of cloth and multiple layers of cloth. Similar transducers are available from other companies.

Ultrasonics could be a valuable sensing technique for automated assembly of flexible materials. One of the difficulties in using ultrasonics is detecting a soft piece of cloth on top of a hard surface, e.g. folding table. Presumably this can be done if the accuracy of the sensor is great enough. Non-contact distance measurement can be by optical as well as ultrasonic means. However, optical distance measurement is expensive (>$2000) and has a shorter range (± 0.5 inch (1.3 cm) typical). Ultrasonic units are somewhat cheaper but still on the expensive side ($200-$500). The expense of ultrasonics is warranted in many situations where non-contact distance measurement is needed.

2.1.5 Capacitive Sensors

Capacitive proximity sensors detect the presence of an object by sensing its dielectric properties. These sensors use an open capacitor in an oscillating circuit. When the object being sensed passes near the sensor, the frequency of the oscillating circuit changes. Figure 2-10 shows typical specification of some industry capacitive proximity sensors.

In principle this form of sensor could be used for detecting the edge of textiles. In practice, however, one or two plies of cloth represents too small a dielectric

CAPACITIVE SENSORS

RECHNER

for the detection of all materials liquid and solid

Supply 10 - 35 VDC

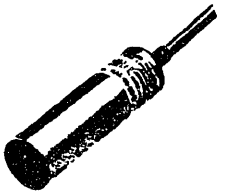
SERIES 70 npn - transistor - output

Eg. KAS-70-20-S

SERIES 80 pnp - transistor - output

Eg. KAS-80-20-S

For S and O types see page 4.
LED Indicator - standard on all sizes.



self contained – no amplifier necessary

Non stock sensors – 6-8 week delivery							
KAS-70-14-S-M12	2	no	12 thread	70	PVC	100	4.1
KAS-70-23-S	2-15	no	20	70	PA	400	1.4
KAS-70-23-S-M22	2-15	no	22 thread	70	PA	400	4.2
KAS-70-41-S	2-40	no	40	70	PA	400	1.8
KAS-70-50-S	2-40	yes	50	70	PA	400	1.9
KAS-70-53-S	2-50	no	50	70	PA	400	1.9
KAS-70-61-S	5-60	no	64	70	PA	400	1.10

Detection Distances are for a steel target 1mm thick, equal to or greater than the sensor diameter.

All brass is chrome plated

Figure 2-10: A Capacitive Proximity Switch.
(From Rechner Electronics Inc. sales literature)

change to be detected. This type of sensor is capable of sensing a stack of textiles but this is not very useful. An interesting footnote is that if the cloth were wet, the sensor could probably detect it.

2.1.6 Inductive Proximity Sensors

Inductive proximity sensors are used for non-contact sensing of metallic objects. The sensors use a probe or core wound with a coil. The coil is driven at a high frequency by an oscillator circuit. When a conductive object passes by the tip of the sensor, eddy currents are produced in the object. This additional load causes the oscillations to cease, and the output state of the sensor to change.

It is obvious that this type of sensor cannot be used to detect most flexible materials. However, it is a very useful for sensing moving parts in automated equipment.

2.1.7 Contact Sensors

The success of any contact sensing method for flexible materials depends on the flexibility of the material. Obviously, if one is working with leather, contact methods are going to be more successful than if one were working with silk. Furthermore, the success of a sensor improves as

its proximity to an actuator increases. For example, consider a piece of fabric moving along on a conveyor belt. The conveyor belt consists of two belts with a 12 inch (30 cm) wide gap between them. A limit switch, which triggers at one gram, is placed right in between the two belt. Another limit switch is placed right near one of the belts. The purpose of these switches is to detect the passage of cloth on the conveyor belt. It should be obvious that the switch right near the belt will distort the cloth less than the other switch, see Figure 2-11.

Most methods of contact sensing of flexible materials are not very reliable. A trip wire or hoop used to detect a cloth edge is likely to be either too stiff, distorting the cloth, or too limp, making it a hair-trigger design. Lint build up and other forms of contamination make the problem even worst. Still, in some applications, contact sensing may prove to be a useful and inexpensive means of sensing flexible materials.

2.1.8 Thermal Sensors

Thermal sensing would make use of the radiant properties of the cloth to determine its position. The author knows of no commercials systems that use this principle.

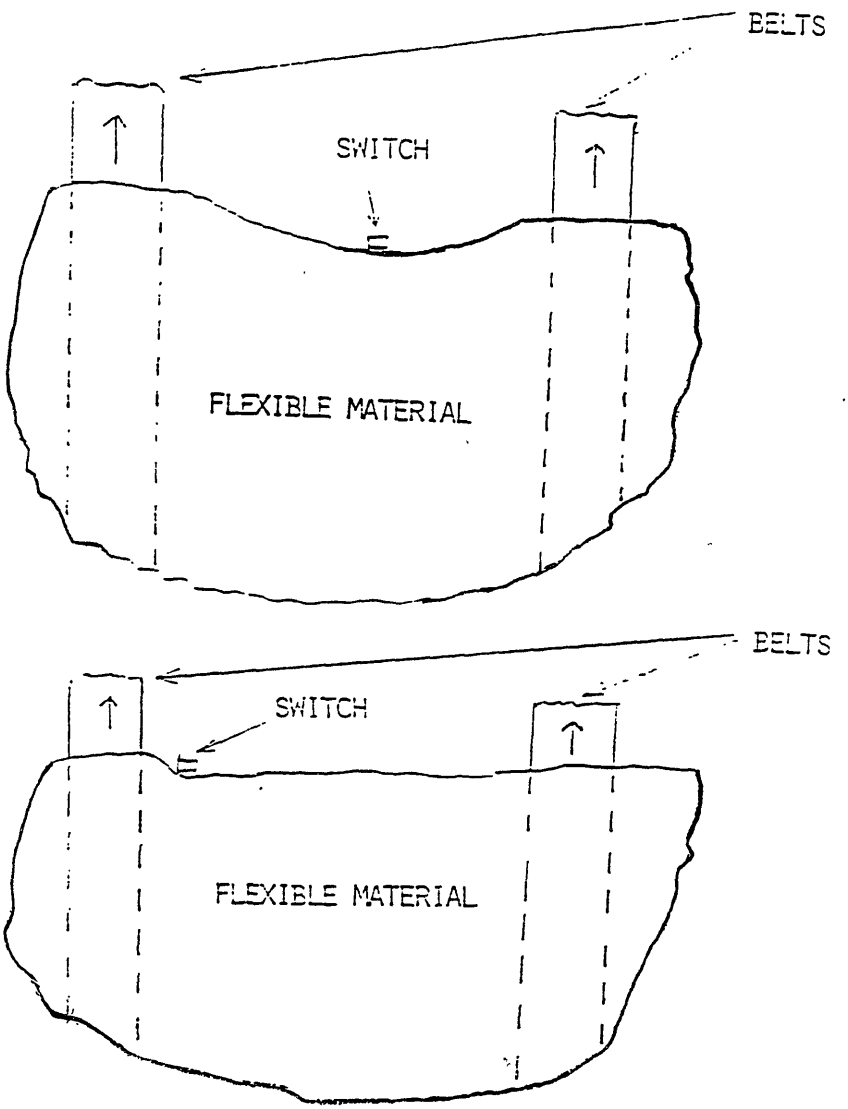


Figure 2-11: A conveyor belt showing the preferred placement of a contact sensor.

2.1.9 Electrostatic Sensors

Electrostatic sensing would make use of the fact that most textiles and other flexible materials pick up a static charge during manufacturing. An electrostatic charge sensor could be used as a proximity sensor for such materials.

2.1.10 Summary

Various types of sensors for flexible materials have been discussed. The advantages, disadvantages, range, accuracy, and/or application of each can be summarized as follows.

1) Photoelectric sensors: fairly inexpensive(a,b)

   a) Thru-beam; long range, accurate, alignment and packaging is difficult.
   b) Reflective; shorter range, accurate, adjustment and packaging are easy, alignment not important
   c) Optical distance measurement; very short working range, very accurate
   d) Vision systems; expensive, complex, wide range of application

2) Pneumatic: inexpensive, could cause fluttering

   a) Back-pressure; must be very close to object, easy to package
   b) Thru-gap; longer range, packaging more difficult
   c) Cross-jet; very long range

3) Ultrasonic:

   Accurate distance measurement.
   Detect the difference between one ply and two plies.
   Long range.
   Moderately expensive.

4) Capacitive:

   Flexible materials too thin for detection.

5) Inductive:

   Used only for metals.

6) Contact:

   Not reliable.
   Deforms the flexible material.

Other possible methods of sensing flexible materials include thermal and electrostatic.

2.2 Cloth Pickers

2.2.1 Introduction

One of the long standing problems with automated apparel manufacturing is finding a good way for the machinery to pick up pieces of cloth. The root of the problem is simply the flexible nature of the cloth. For a comparison consider a robot picking up a block of steel versus a robot picking up a suit back. For the block of steel, a simple two finger end-effector will suffice. Once the robot has picked up the block of steel, the location of all points on that block are known or can be calculated. If the same end-effector is used to pick up the suit back, the cloth part will simply drape like a magician's handkerchief. About the only thing that is known is where the point of pick up is. An attempt to put the cloth down on a flat surface would be a futile exercise. What if the robot picked the piece of cloth up at two points? In this case, some shapes and sizes of cloth could be picked up and put down flat, but most could not. The subject of how many pickers are needed to pick up a given piece of cloth is called **The multi-point pick-up problem.** This discussion will be limited to how different types of single point pickers work.

A number of commercial and experimental cloth pickers are available for development work. Most of these use some form of mechanical pick-up, although vacuum suction, adhesive tape, electrostatic attraction, and many other lessor known techniques. Some of the requirements for a good picker is: 1) reliability, 2) works over a wide range of different materials, and 3) does not damage the cloth during pick-up. More practical issues such as cost and packaging must also be considered for an actual commercial system. To show how these pickers work, the following section will discuss a number of the "better" pickers.

2.2.2 Walton Pickers

One of the more commercially successful pickers has been the Walton picker. The picker was developed by an inventor by the name of Walton and later improved by a designer named George Wood. The pickers can be bought for about $15 each, which is remarkably cheap compared to other pickers. The end of the picker was originally made from two short lengths of hacksaw blade, see Figure 2-12. An air cylinder is arranged so that when it is activated the two pieces of hacksaw blade come together with a shearing motion. This shearing motion locks the hacksaw blades into the threads of the fabric, making a fairly secure "grip." The pickers are small, approximately 0.5 inch (1.3 cm)

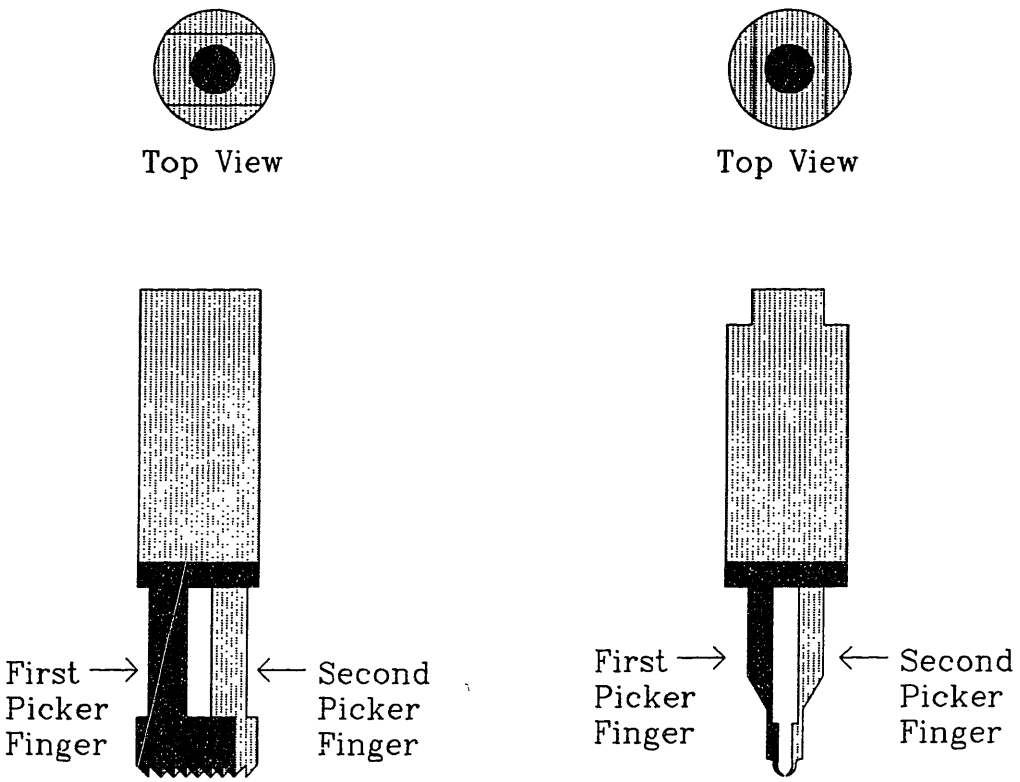


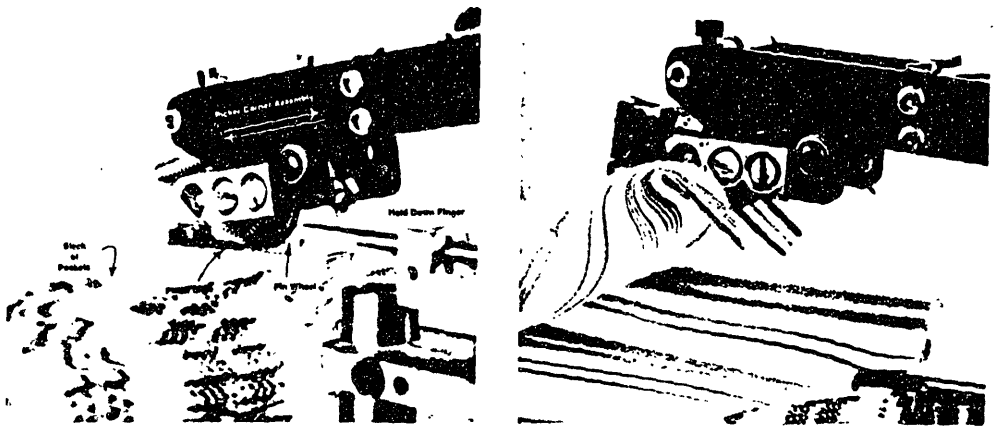
Figure 2-12: Walton Cloth Picker used by Draper Labs.

cylinder by 4 inches (10 cm) long. They are very reliable and usable over a moderate range of fabrics. They do have a tendency to mark the fabric and could not be used on fine materials.

The Walton pickers are used by both Draper Laboratory and Singer Corporation. Draper Labs has used them for several years on the $(TC)^2$ program and was involved in improving their design. Singer uses the pickers on a number of their automated sewing systems. Through years of experimental use, it has been found that the pickers are well suited for commercial applications.

2.2.3 Clupickers

Perhaps the biggest commercial competitor with the Walton picker is the Clupicker made by Jet Sew. Jet Sew is a division of Arrow Shirt Co. The Clupicker is a clever device that reminds one of a Swiss watch. The picker consists of a small thumbwheel and a "finger" , see Figure 2-13. To pick up the cloth, the thumbwheel is turned in one direction so that the cloth is forced up into the finger. To release the cloth the wheel is turned in the opposite direction so that the cloth is forced out flat. The actual operation of the picker is more complex than just described. The Clupicker seems to be very reliable although some people



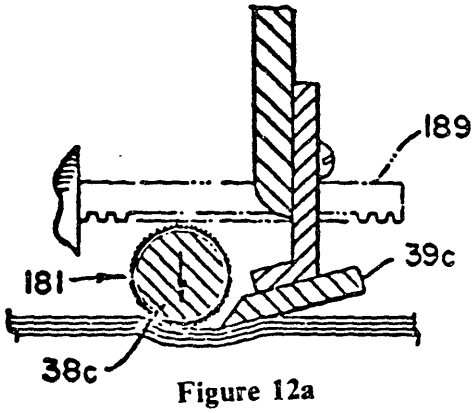


Figure 12a

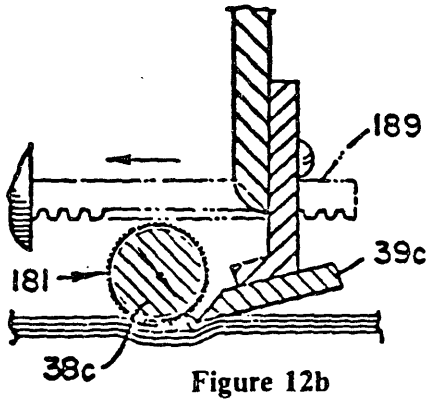


Figure 12b

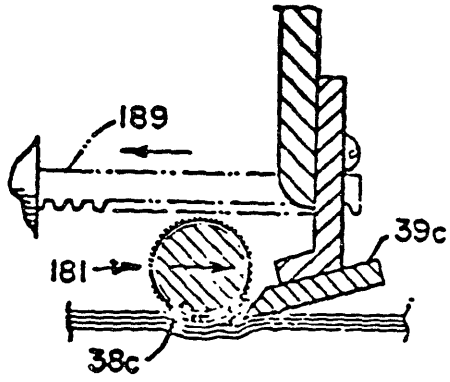


Figure 2-13: Clupicker made by Jet Sew Inc.

claim otherwise after having seen it fail during demonstrations. (The author watched some Clupickers in action for about 15 minutes at the 1986 Bobbin Show and never saw a missed pick.) One of the problems with the Clupicker is the cost. Although they are not readily available, a Jet Sew salesman figured that the Clupickers would cost about $450 each to make and would sell for $900 each.

Picking up cloth using a Clupicker is fundamentally different from using most pickers. The Clupicker makes a fold in the cloth along the edge where pick up is to occur. This fold has a number of effects. First the fold has the effect of stabilizing the cloth piece. The fold line makes the cloth effectively stiffer allowing the use of fewer pickers and improving reliability. Secondly, the fold could crease some fabrics and distort the cloth piece. The cloth is forced into a complete 180 degree bend, hence some types of cloth are bound to crease. Lastly, the fold makes it hard to pick up oddly shaped pieces of fabric. Jet Sew uses the Clupickers on rectangular pieces of cloth, mostly pockets, cuffs, etc. It is suspected that the reliability of the Clupicker is a function of the cloth weight or thickness and proper adjustment of the picker. Not having worked with one, it is impossible for this author to know how difficult the Clupickers are to setup.

2.2.4 Polytex Pickers

Polytex pickers are a Swiss made cloth pick up device. They pick up the cloth using two pairs of curved needles, see Figure 2-14. The picker is basically a small mechanical module 1.5 x 1.5 x 5 inches (4 x 4 x 1.3 cm), weighing 0.25 kg, with 4 curved hollow needles in the base, crossing each other in pairs. The needles project when pressed onto the top ply, actuated by compressed air, achieving the pickup by penetration of needles into fabric; then compressed air is blown through the needles to separate the ply. Retracting the needles releases the ply of fabric. The depth of penetration of the needles is between 0 to 15 mm, and is adjusted for different fabrics. The manufacturer claims that the picker will not mark delicate fabrics. One Polytex picker costs about $250.

Polytex pickers have a number of advantages over other types of pickers. First and foremost is the ability to blow air through the needles to help separate the top ply from the bottom ply. This ply separation is a big problem for most pickers, although no one, including Polytex, has come up with a foolproof solution. Another advantage of the Polytex picker is its ability to withstand fairly high pulling forces on the cloth without dropping it. A

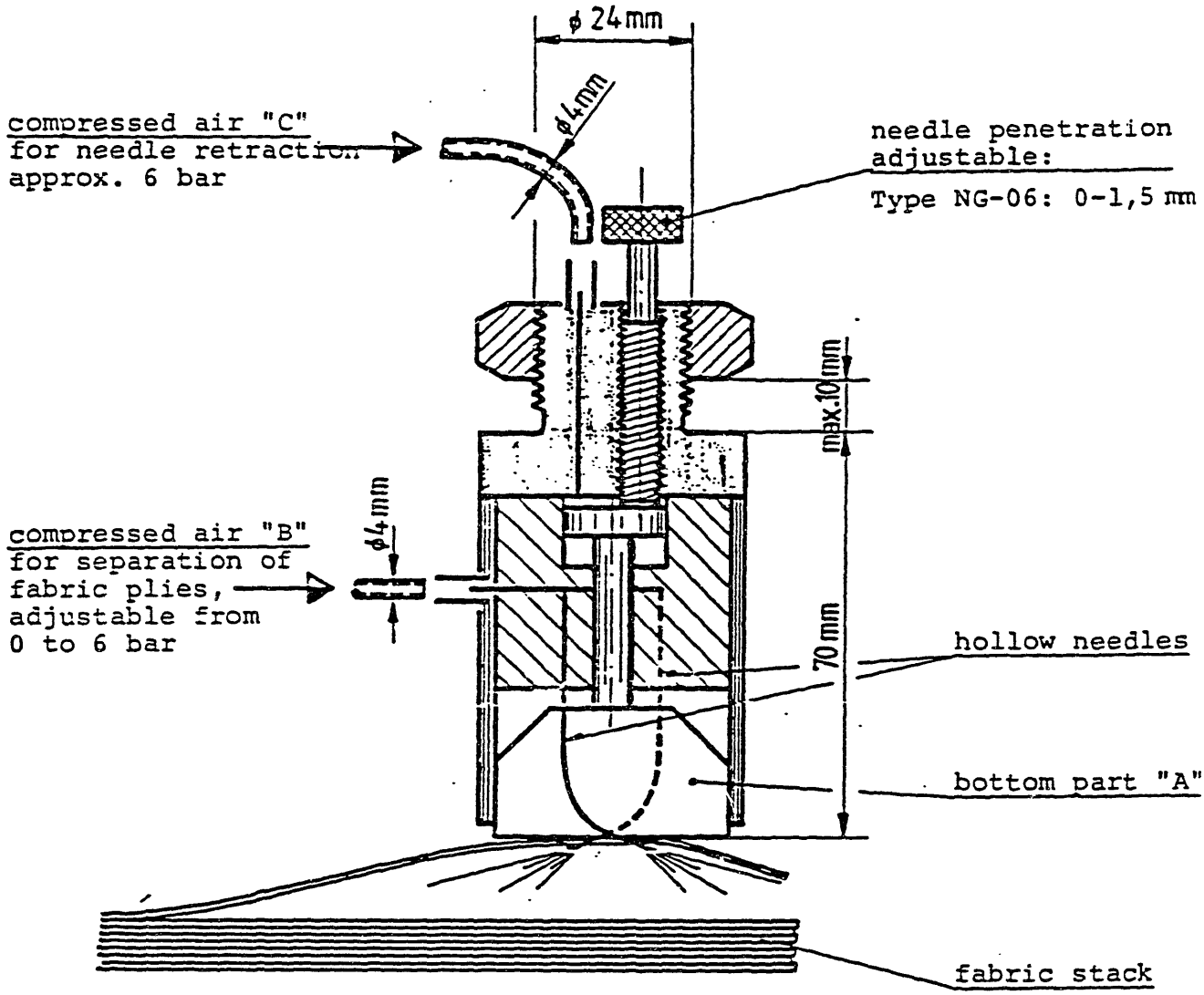


Figure 2-14: Polytex Cloth Picker.

disadvantages of the Polytex picker is that it might rip or deform the cloth if too high a load was on a pair of needles. Another possible problem is the relatively large size and weight of the device. The size makes it impossible to pack a large number in a small place. The weight is undesirable on a commercial robot. It is this author's belief that both the size and the weight could be reduced because they are not inherent to the design. One final problem is the depth of needle penetration must be adjusted each time the pickers are used on different thicknesses of fabric. This penetration must be set carefully. Setting too deep a penetration might cause the picker to pick-up more than one ply at a time. Overall the Polytex picker is works effectively.

2.2.5 MARS Pick-up Device

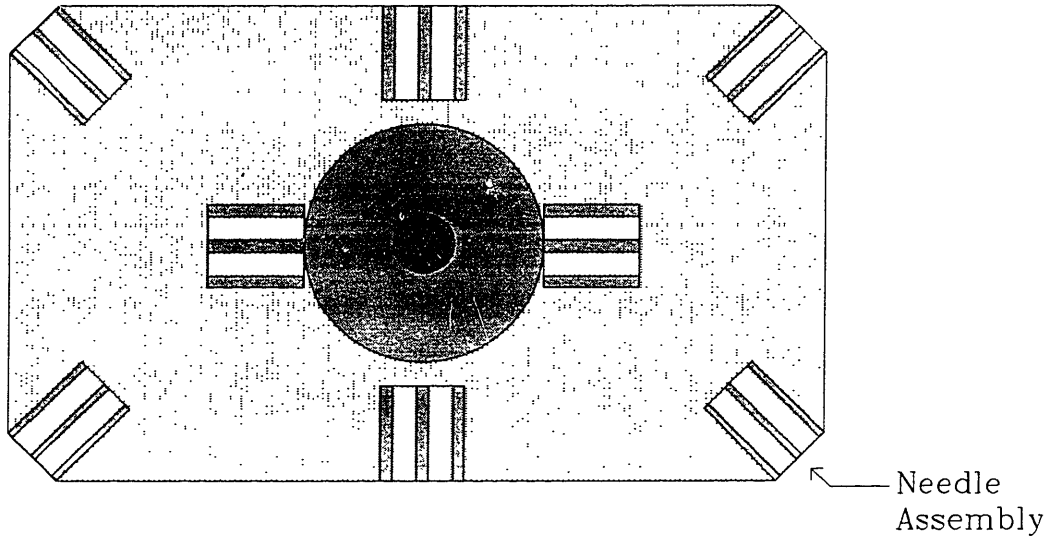
Although it is not a true single point picker, the MARS pick-up device will be discuss here. The MARS manipulator was designed by Singer Co. to work with their MARS Cartesian coordinate robotic sewing systems. The MARS systems are capable of picking a piece of cloth off a stack and then sewing edging on the piece. The manipulator uses needles similar to the ones used in the Polytex pickers. In the MARS manipulator, however, the manipulators are custom made and the needles are placed around the perimeter, see Figure

2-15. This provides good control of the whole piece of fabric at the expense of system flexibility. The MARS pick-up devices appeared to be only moderately reliable. During a demonstration, pick-ups were missed about 5% of the time, apparently caused by incomplete needle penetration. The MARS manipulators are of extremely limited use. For a discussion of a similar design see Parker[3]

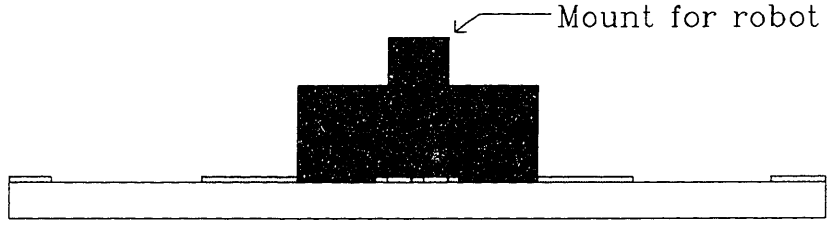
2.2.6 Vacuum Pickers

Perhaps the most obvious solution to the cloth pick up problem is the vacuum picker. In this method, the free end of a vacuum hose can be used to pick up a piece of cloth. Although vacuum pickers work very reliably on a single ply of cloth, they are all but useless when trying to pick up the top ply of a stack of fabric. The same suction that picks up the top ply also picks up one or two more plies. Vacuum pickers are extremely useful on non-porous materials such as paper. They are used to pick up labels one at a time on a commercially available sewing work station. A mechanism sucks up one label off the top of a stack and delivers it to the seamstress. Vacuum pickers have only limited application in apparel manufacturing.

One novel approach to the vacuum picker problem was discovered by Ann Ito[4]. She discovered that inducing a forced vortex flow field around the outside of a vacuum



Top view



Side view

Figure 2-15: MARS Pick-up Device (Singer)

picker caused the top ply of fabric to lift up without disturbing the rest of the stack. The picker consisted of two concentric cylinders, see Figure 2-16. The outside cylinder supplied air in a vortex flow pattern which was induced by a spiral groove in the outer cylinder. The inner cylinder was connected to vacuum. By adjusting the relative strengths of the vacuum and air pressure, the picker could achieve acceptable performance. Since the picker was done as a Bachelor's Thesis it has not been taken to completion. Further testing for reliability is needed. The Ann Ito vacuum picker shows quite a bit of promise.

2.2.7 Adhesive Pickers

The use or attempted use of tape and adhesives to pick up pieces of cloth has been going on for many years. The concept is straight forward; put something sticky on the end of your pick up device, press it down on the cloth piece, pick up the cloth piece, then use a stripping mechanism to release the cloth. This concept has several problems. Anyone who has ever stuck a piece of tape to their clothes knows that after a few repeated applications the tape is useless. The lint from the cloth sticks to the tape. This problem can be overcome by using a new piece of tape each time but that costs money. Most adhesive pickers use a self-advancing roll of tape to pick up the cloth. Another

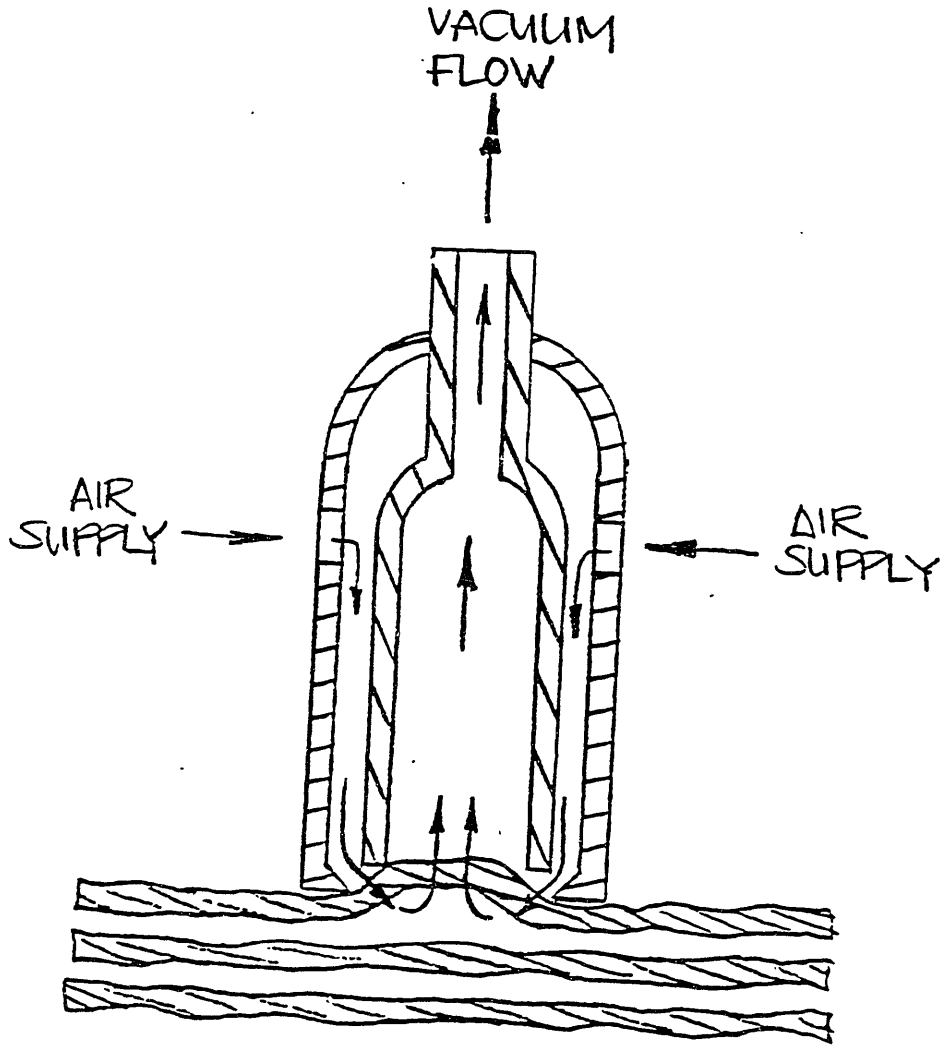


Figure 2-16: Vortex Vacuum Picker Designed by Ann Ito.

problem is that the tape must not damage the cloth. If the tape is too strong it will ruin the surface of the cloth. The adhesive picker must not leave behind any adhesive. Some types of cloth with textured or abraded surfaces will not work at all with adhesive pickers because the picker simply pulls the lint off the surface and cannot grab the body of the cloth. Adhesive pick-up does have the advantage of being an "area" pick-up. This means that the pick-up force is distributed over an area (unlike needle pick-up devices that only contact a few threads). This leads to less locale deformation of the cloth.

One experimental adhesive picker was developed by Parker[3]. They found an adhesive tape that worked well. The strength of the adhesive bond drop to about 50% of the original strength after about 25 cycles. The picker was simply a spool of tape guided around a series of pins, see Figure 2-17. They claimed that the adhesive picker had "virtually no chance for acquiring more than one ply at a time." This is somewhat doubtful because the pick up device employs no active ply separator. Durkopp employed a very similar adhesive picker on one of their pocket welt machine, see Figure 2-18. They used a 2 inch (5 cm) wide roll of tape. The adhesive picker worked very well for that particular application.

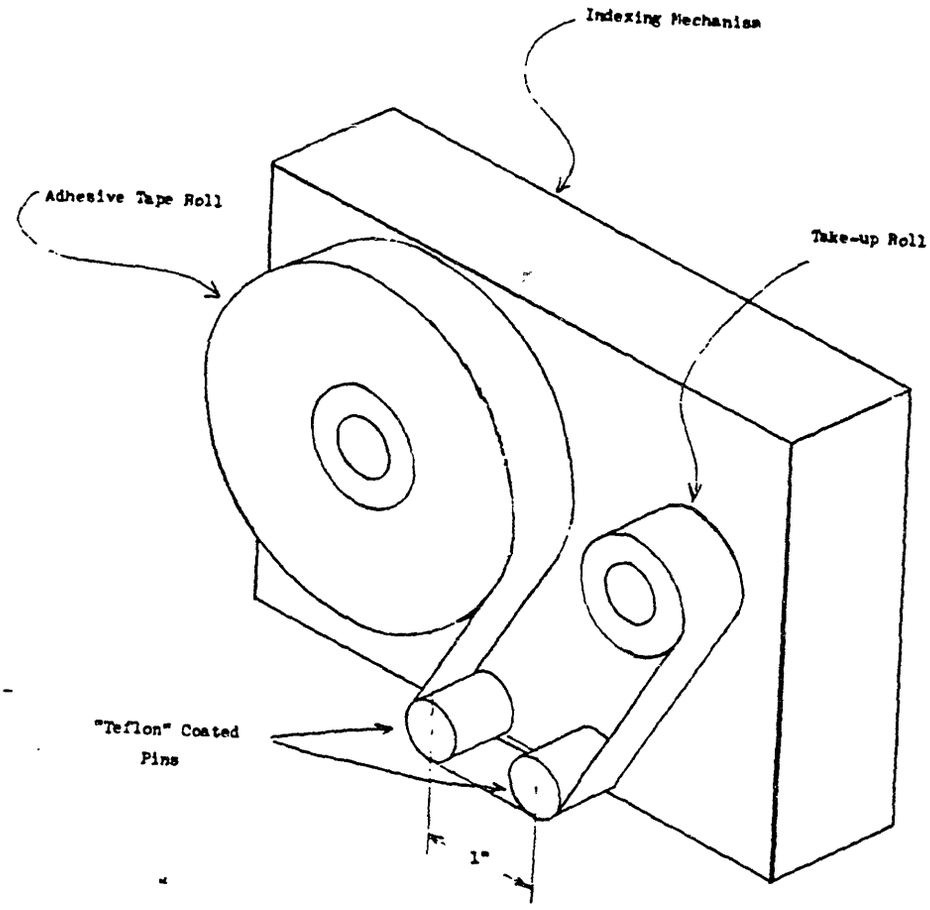
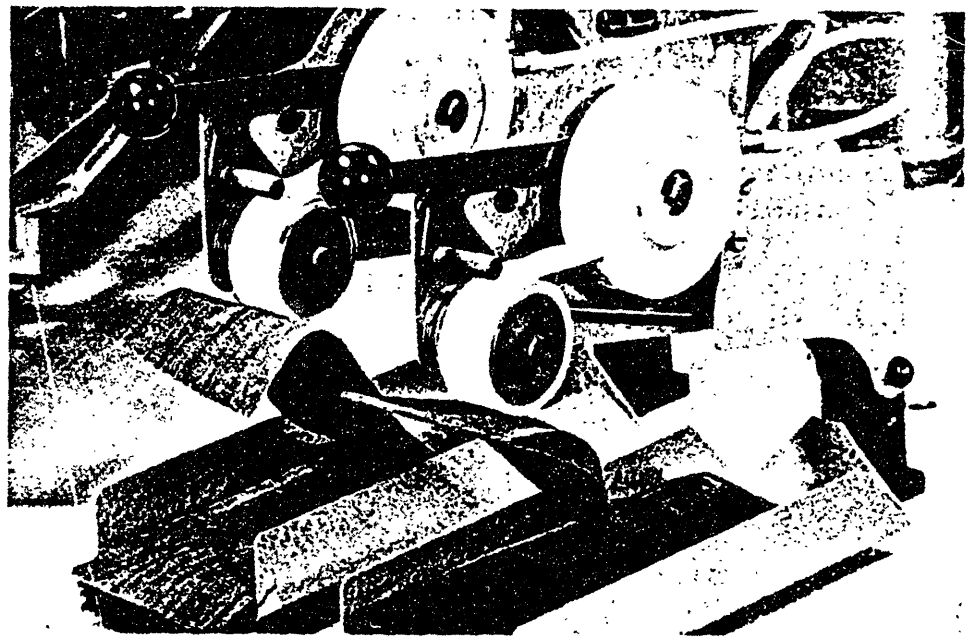


Figure 2-17: Experimental Adhesive Picker.

DURKOPP ADHESIVE PICK-UP ON POCKET WELT



Durkopp adhesive pick-up used on pocket welt machine.

Figure 2-18: Durkopp Adhesive Picker.

2.2.8 Other Pickers

The above discussion of pickers is necessarily incomplete. The pickers chosen for discussion are either successful, popular, or have received attention in the literature. A great number of cloth pick-up devices have been invented and patterned over the years. Murray[5] performed patent searches and interviewed manufacturers of textile equipment to accumulate information about cloth pick up devices. Murray's paper presents a list of about 130 patents from 1914-1975. The article also includes many pictures of the pick up devices. Furthermore, he discusses the secrecy with which the apparel industry tries to hide "proprietary equipment." This discussion points out the need for cooperation among members of the apparel manufacturing community. It is worth noting that although many pickers exist, most are functionally similar to the pickers described above.

2.3 End-Effectors for Flexible Materials

2.3.1 Introduction

Unlike sensors and pickers, there are very few end-effectors for cloth. $(TC)^2$'s sleeve machine has just about the only end-effector in use. There are some machines that use a straight line of pickers on a movable bar. This could be considered to be an end-effector but there is really nothing to be discussed. Hence the discussion of end-effectors will be limited to the one on $(TC)^2$'s sleeve machine.

2.3.2 $(TC)^2$'s end-effector

The end-effector for the $(TC)^2$ sleeve machine is call the *spline*. It consists of a set of 12 pickers mounted on a flexible member that can be bent into a curve, see Figure 2-19. The spline shape is controlled by two stepper motors. The spline is held by three rigid sections. The stepper motors can move the two end sections relative to the center section in order to bend the spline, see Figure 2-20. Due to the way that the spline is supported, it is bent into a calculable curve. In order to pick up the inseam or outseam of a sleeve, the spline is bent into a curve that closely matches the curve of the inseam or outseam. Although this end-effector works well for the curved section of the

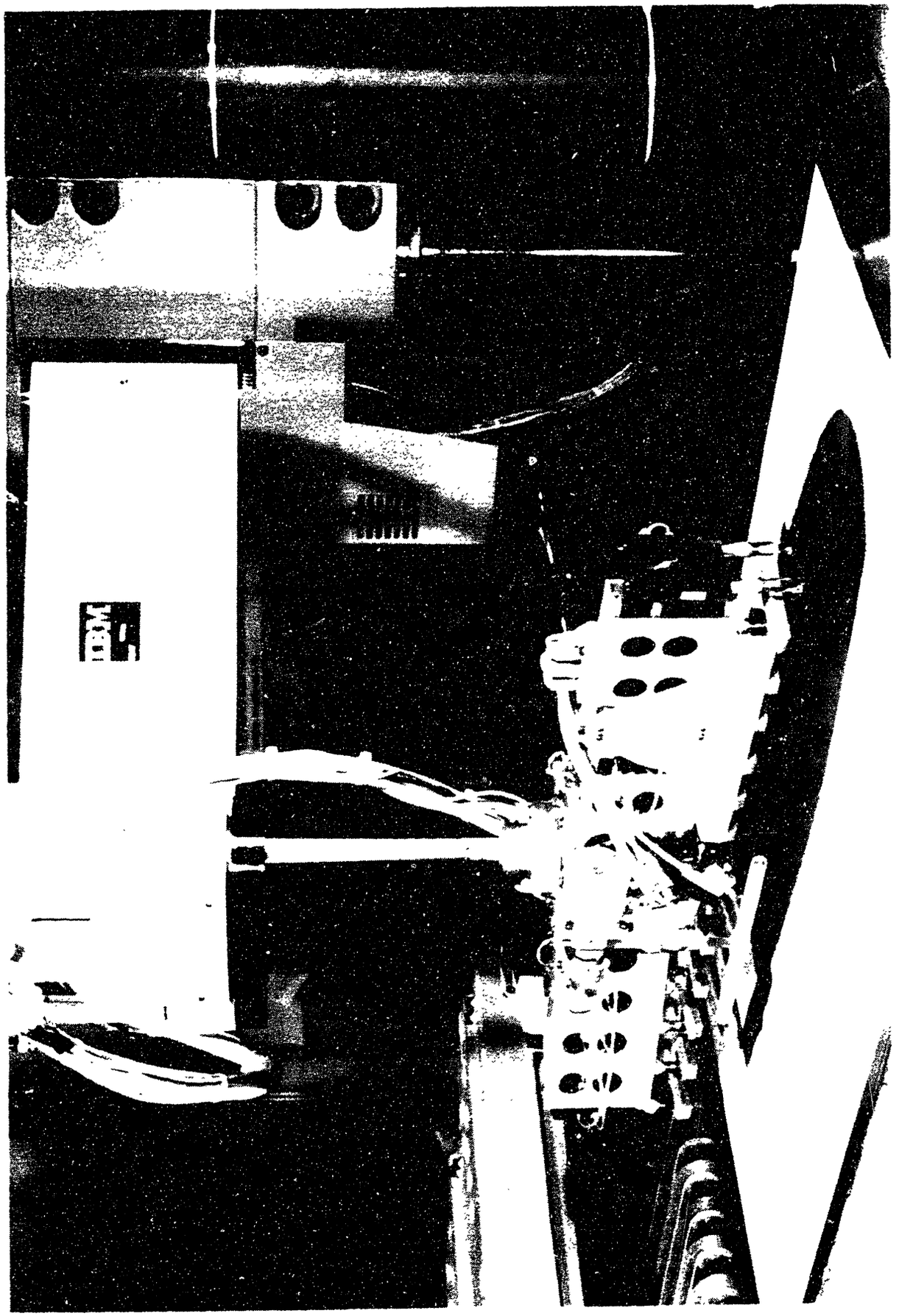


Figure 2-19: $(TC)^2$ End Effector for Sleeve Machine.

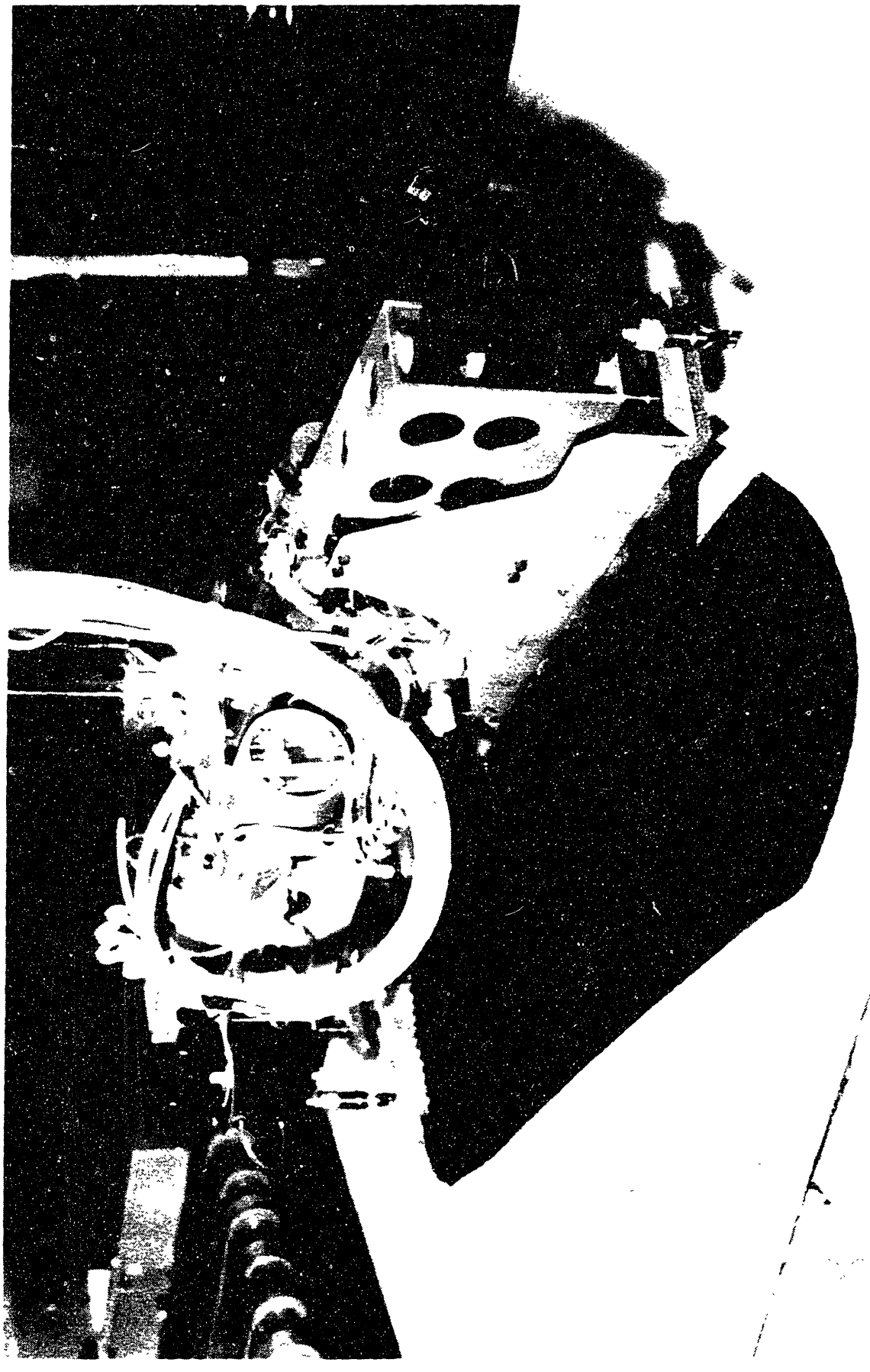
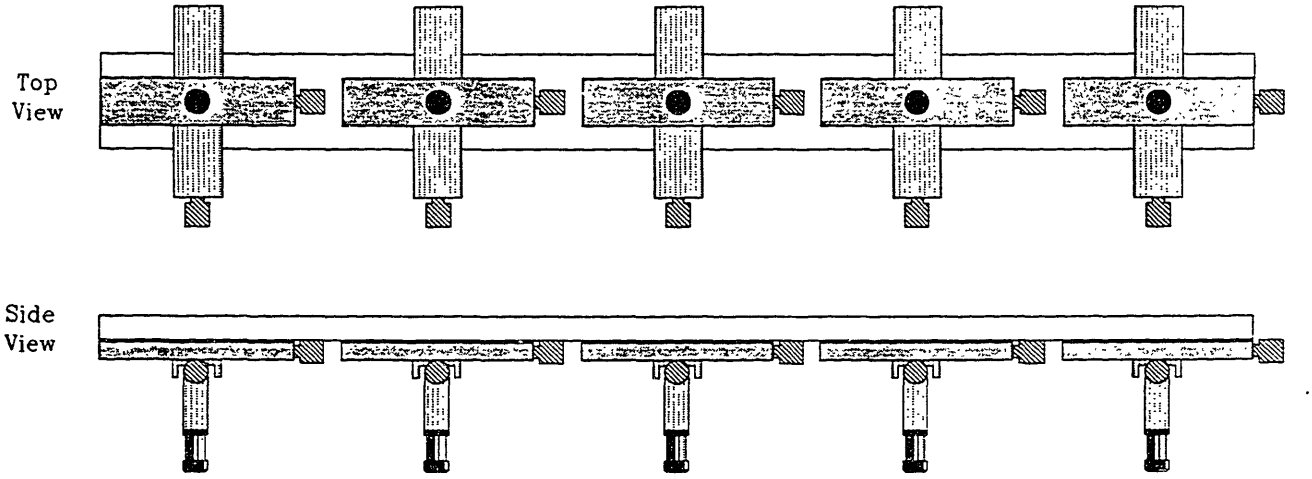


Figure 2-20: Top View of $(TC)^2$ End Effector.

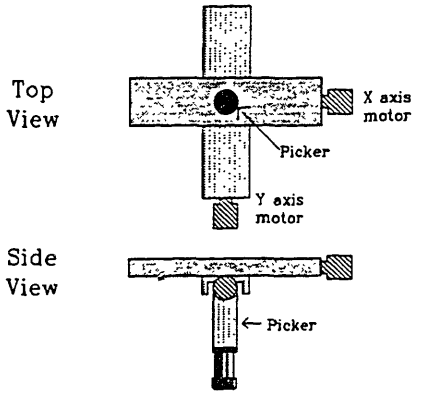
sleeve, it fails to adequately support the vent (cuff) area of the sleeve. To alleviate this problem a number of additional pickers were placed down at one end of the spline, see Figure 2-19. This was a cheap, functional solution to the problem of picking up a sleeve. For an end-effector to be really versatile, it must have greater flexibility than the spline.

2.3.3 Proposed Flexible End-Effector

There is a very simple alternative to the $(TC)^2$ end-effector. The proposed end-effector would consist of a line of 6 to 8 pickers spaced 3 to 4 inches (7.6 to 10 cm) apart, see Figure 2-21. Each picker would be mounted on a two axis (X-Y) slide driven by two stepper motors. Each picker could be placed anywhere within a 3 to 4 inch (7.6 to 10 cm) square area. The proposed end-effector would be capable of picking up a wide range of shapes. If one area, like the vent, was difficult to control, then two adjacent picker could be moved close together to handle it. One problem with this type of end-effector is the cost. The increased flexibility would probably outweigh the increased cost. This proposed end-effector is needed to place pickers in better locations for proper cloth control.



Sketch of end effector with five pickers.



Picker positioning
sub-assembly

Figure 2-21: Proposed Flexible End Effector

3 Automated Assembly of Men's Dress Pants

3.1 Introduction

This chapter demonstrates the process of assembling a pair of men's dress pants, both manually and automatically. It is based on Simplicity pattern number 6668, view four. The goal is to explore the feasibility of appling $(TC)^2$ technology to the automated sewing and folding of these pants. It should be noted that the current machine is not capable of "making" these pants. This is not to say, however, that the technology is not capable of "making" the pants. If the technology is not capable of performing a specific operation, then it will be clearly indicated.

The current $(TC)^2$ machine was designed to manufacture men's suit coat sleeves. This goal led to certain specific hardware elements that are not particularly general purpose. Most notable is the end-effector of the robot. The current end-effector, called the spline as described in the last chapter, has about ten pickers in a line with 2 inch (5 cm) spacing between pickers. This line of pickers can be bent into a curve to roughly match the curve of the suit sleeve. The spline works fairly well on suit sleeves but it is not particularly useful on other shapes. In fact, to make it work with suit sleeves, it was necessary to add a number of

additional pickers on one end of the spline to pick up the vent (cuff). Another type of end-effector could have less pickers and work just as well or better.

Although automated assembly includes both folding and sewing, the focus of the current study is folding. The reason for this is that the folding or positioning of the cloth before it is sewn is known to be the most difficult part of automated apparel manufacturing. The current sewing module configuration is very flexible. It is capable of sewing circles, squares, or most any shape. The folding station, on the other hand, has had much difficulty working with relatively simple shapes. As previously stated, one of the problems with folding is the robot end-effector. It limits picker placement to a curved line. For the current purposes the existence of a very flexible end-effector will be assumed, see section 2.3. The reason for assuming the use of this proposed flexible end-effector is that it will allow the placement of pickers anywhere that they are needed. If the use of the $(TC)^2$ end-effector was assumed, care would have to be taken that the mechanical limits of that end-effector are not violated. All end-effectors have limits to where they can place pickers, but one could easily be built with much less limitation than the $(TC)^2$ end-effector.

3.2 Comparison of Automated and Manual Pants Assembly

The first step in developing an automated pants assembly line is to study the manual assembly procedure. The manual assembly procedure has 54 steps. The order of these steps is not etched in stone. For example, the front pocket can be sewn on first and then the back pocket, or vice versa. Some of the procedures, however, must be performed in the order given. For example, the front pocket must be sewn to the pants front before the side seams are sewn. A detailed description of the manual assembly procedures is given in Appendix A.

The steps for making the Simplicity #6668 pants were written in consideration of the easiest way for a person to assemble the garment. However, the easiest way for a human to assemble a garment is not likely to be the easiest way for a machine to assemble the garment. Hence, the order and method of each step must be modified for use with an automated sewing system. All of the modifications to the assembly procedure will be targeted to: 1) reduce the number and cost of sewing stations, and 2) reduce the time to manufacture the garment. The are several ways to achieve these objectives:

1) Combine assembly steps so that more foldirg and sewing can be done at each pair of folding and sewing stations.

2) Perform multiple "unrelated" folding operations at folding stations when possible.

3) Modify the apparel design so that fewer work stations are needed to make the garment.

4) Modify the apparel design so that more of the assembly process can be automated.

5) Arrange the steps that cannot be automated so that they do not break up a number of automated steps. "Hand operations" should be placed at the end of the assembly process whenever possible.

Items 1, 2, and 5 will be demonstrated in the automated pants assembly line. Items 3 and 4 will be discussed in the next section

3.3 Design for Automated Assembly

Large gains in automated manufacturing of apparel can be achieved by working with the apparel designers to design garments that are easily automated. Much of automated sewing is concerned with areas of the garment that the consumer never sees - the seams. Modifications in the shape and type of seams can make a garment much easier to manufacture, yet the end product is functionally, if not visually, unchanged. By forcing the apparel designer to consider the method of manufacturing, the whole process of automated sewing will become more integrated. This type of thinking is the same process that all engineers (designers) must go through in the design of mechanical components.

They must decide whether to make a part by sheet metal bending, casting, extrusion, or some new process. If automated apparel manufacturing is going to succeed, then apparel designers will eventually have to start doing the same thing that mechanical designers have always done. To do so, apparel designers have to develop an understanding of how the design affects automated construction. Once the apparel designers have done what they can from the design stand point, it is then up to the manufacturing engineers to make the best use of the automated textile equipment.

3.4 Layout of an Automated Factory

Work station timing and utilization is important for economical automated manufacturing of apparel. If the work stations are attached in a serial (straight line) fashion, then the production time for each station should be about the same. This means that even if several operations can be done at one station it might be advisable to use two stations. If too many operations are done at one station, it may cause a bottle neck. Factory layout and process planning techniques developed to solve these problems for typical assembly lines are applicable to sewing assembly lines as well. It should be noted that a serial line is not the most efficienct type of assembly line. A discussion of assembly line efficiency is beyond the scope of this work.

Assembly plant floor layout must be arranged so that the right work station can receive the parts it needs at the proper time. Though there are may ways that this can be done, only one will be shown here, see Figures 3-1a-c. As discussed earlier, fully automated weaving, spreading, and pattern cutting is possible with state of the art equipment. Hence the automated sewing line shown here begins just after the cutting table. Since this assembly line is for pants, there will be two serial lines in parallel, one line for the left pant leg and the other line for the right pant leg. Between these two lines is a conveyor belt for transporting the pants pieces from the cutting table to the folding stations. At the end of the conveyor belt is a waste bin for the scrap fabric left over from the cutting. If a folding robot needs a part for its operation it will pick it up off the conveyor belt and put it on the folding table. Figure 3-1 shows the number of the folding/sewing station on the left and the operations to be performed at these stations on the right. At the end of the automated line will be a number of seamstresses who will perform the operations that cannot be automated, e.g. crotch seam and waistband. The last operation is to attach the belt loops and then the pants are finished.

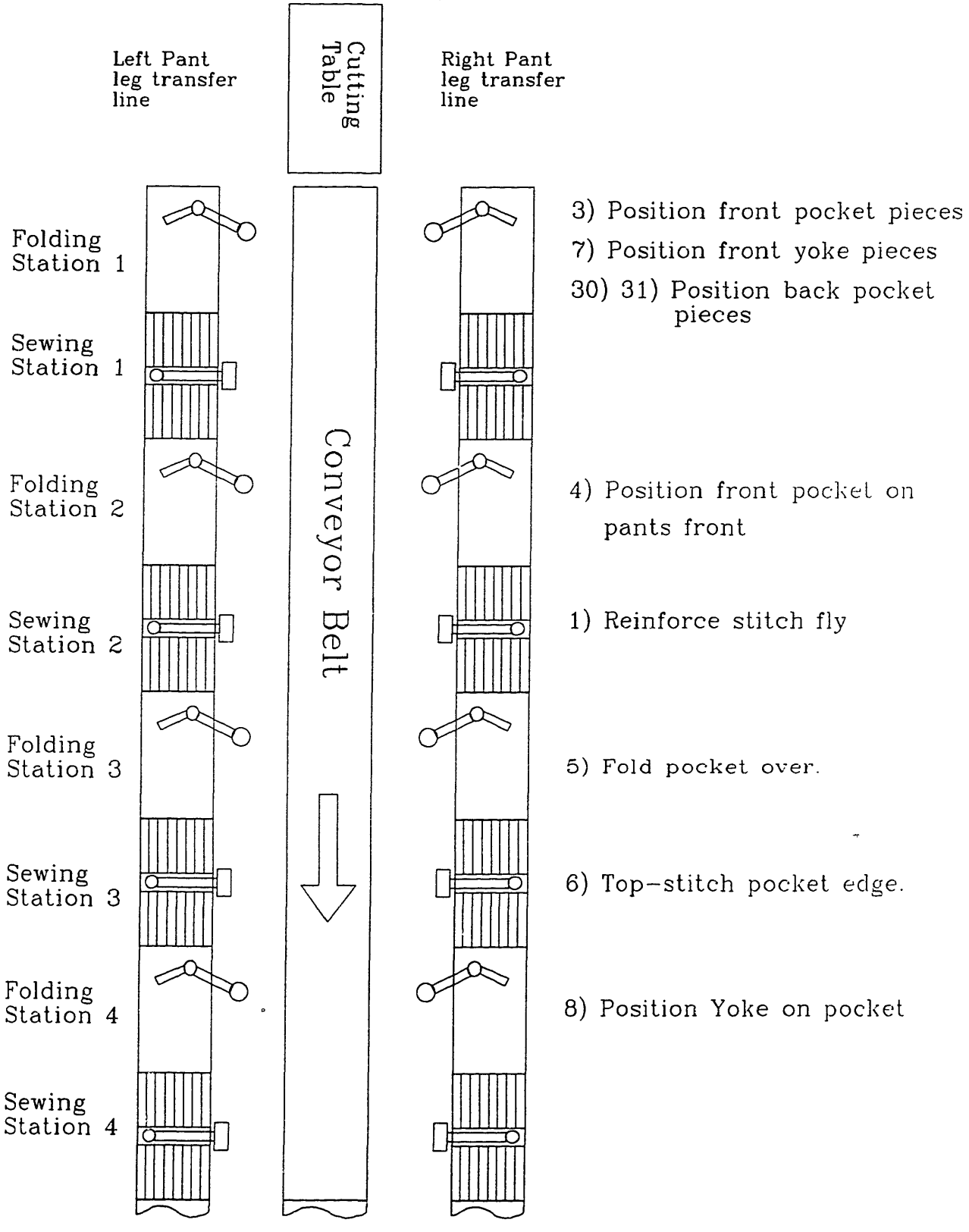


Figure 3-1a: Automated pants assembly line, stations 1 to 4.

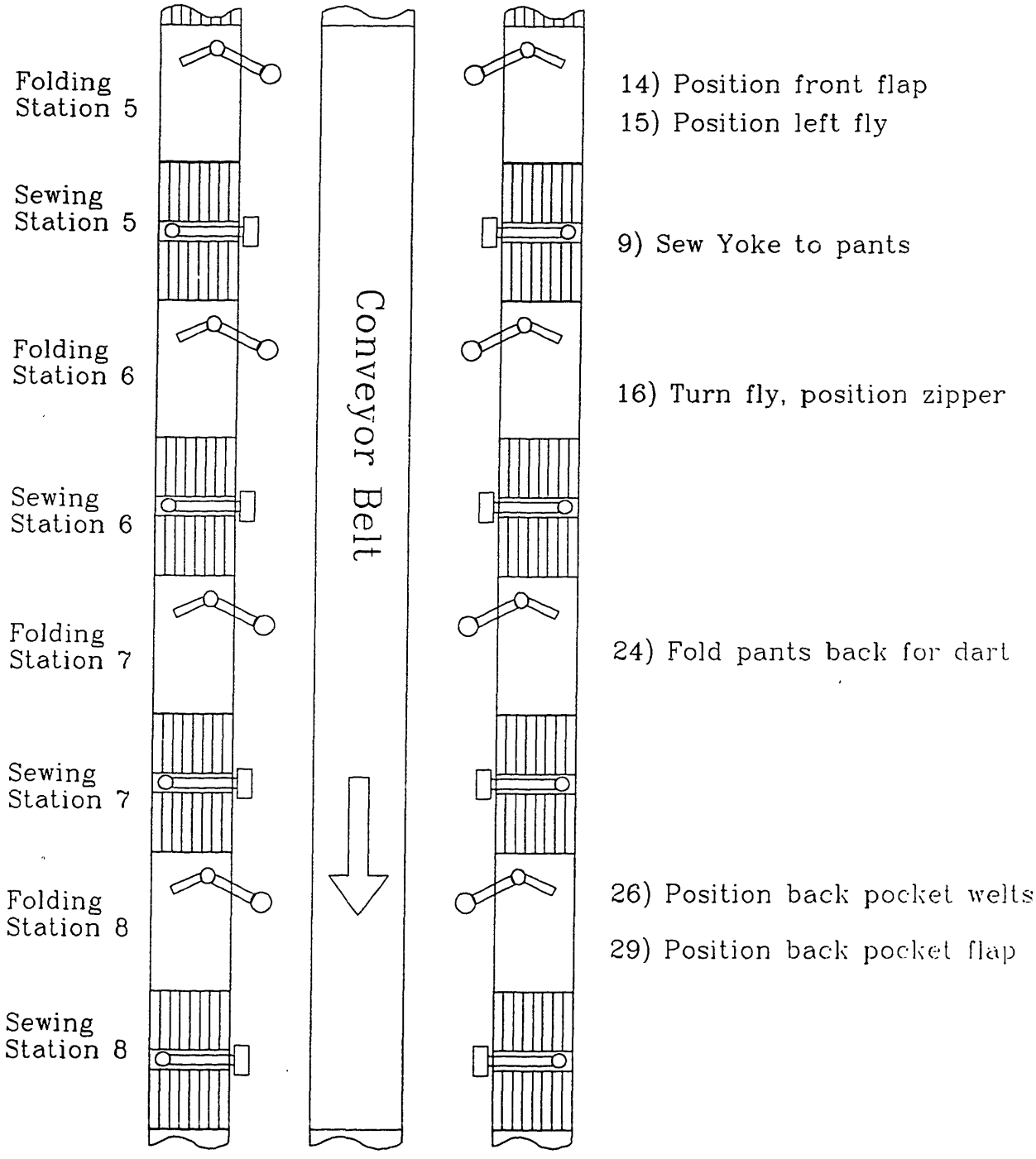


Figure 3-1b: Automated pants assembly line, stations 5 to 8.

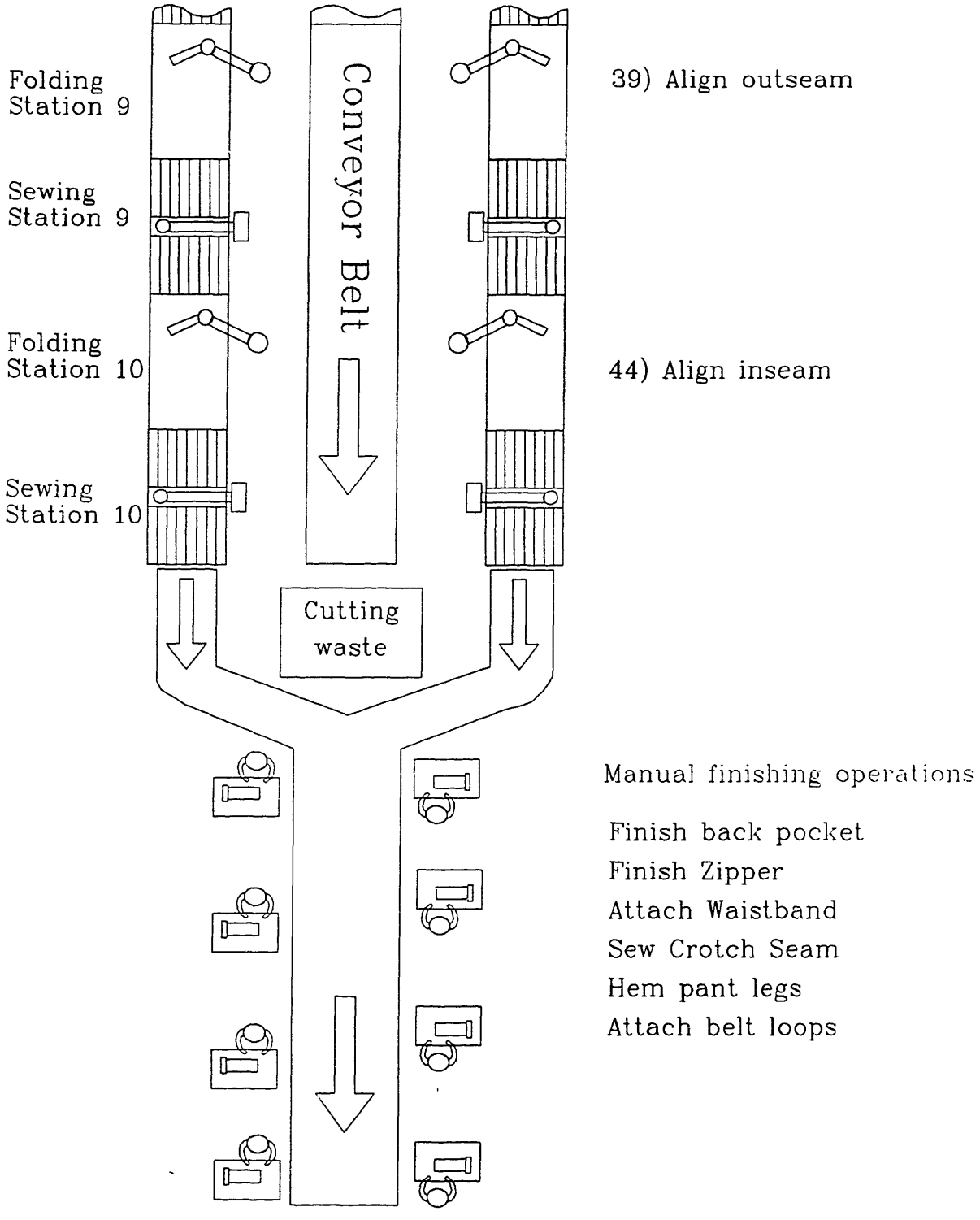


Figure 3-1c: Automated pants assembly line, stations 9 to 10 and hand finishing.

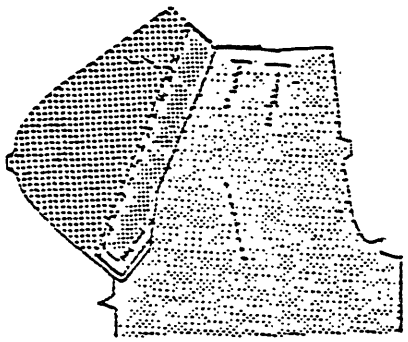
It should be noted that many of the folding/sewing stations are replacing two or three manual operations. This is one of the benefits that automated sewing has over manual sewing. A good example of this is folding/sewing station 5. At the folding station, the front pocket flap and the fly are both positioned. At the sewing station the yoke, front pocket flap, and fly are all sewn to the pants front at once. This prevents sewing at the same place twice for the fly and the yoke, and the pocket flap and the yoke. It would be difficult for a seamstress to hold all four pieces together while sewing them.

This automated sewing assembly line demonstrates how a number of $(TC)^2$ machines can be linked together effectively. This particular assembly line is for pants but there is no reason that it would not work equally well for shirts or jackets. By comparing the order of operations for hand sewing and automated sewing it should be clear that the machine does not try to mimic conventional sewing methods. The automated assembly procedure is shown in detail in Appendix A.

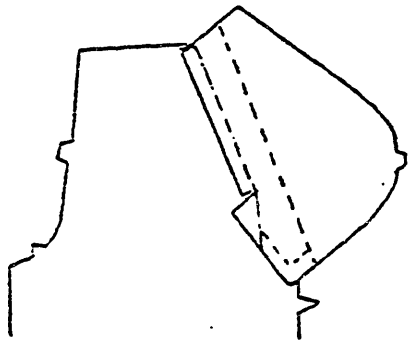
This list of steps in Appendix A is not complete, there are six steps missing. The missing steps are for attaching the interfacing to the front flap, back flap, and to the

waistband. These steps were omitted because they cannot be made automatically the way the parts are designed. They can, however, be made automatically with only slight design modification. The pocket flap design calls for sewing the flap to the interfacing and then turning the two inside out. If this was changed so that the flap was made like a patch pocket, i.e. edges folded under, automation would be easy. In fact, automated patch pocket setters exist and are quite common. As for the waistband, it could be made on a modified collar sewing machine. A couple of companies make automatic collar machines that could work very easily for waistbands. These specialized automatic machines would have to be worked into the assembly line. They would have to get the parts from the cutting table, make the flaps or waistband, and send the parts to the appropriate folding station.

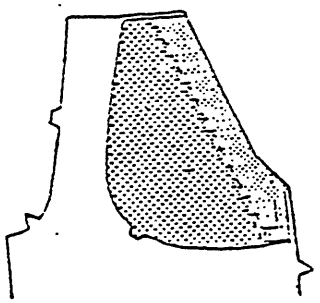
To demonstrate the differences between automated and manual assembly, folding/sewing station 3 will be discussed in detail, see Figure 3-2. At the previous station the front pocket was sewn to the pants front with the right sides together. Now the pocket must be folded over and the pocket edge top-stitched. It is simple to perform this operation manually. To do it automatically, however, requires quite a bit of thought. The reader should keep in



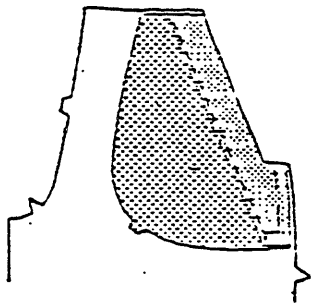
5a) Flip out pocket



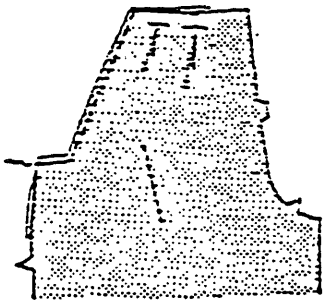
5b) Turn assembly over



5c) Flip pocket over



5d) Fix fold



6) Top-stitch pocket edge

Figure 3-2: Folding/Sewing Station 3, sewing front pocket edge.

mind that the pants pieces must lie flat on the folding table during automated folding. The author developed the following folding sequence for this procedure. The inside edge of the pocket is picked up and moved to the left, step 5a in Figure 3-2. To put the wrong sides together, the assembly must be turned over. The left edge of the assembly, as it appears in step 5a, is picked up and dragged to the right, thus flipping the assembly over, step 5b. The pocket is then picked up on the right hand edge and moved to the left and positioned on top of the pant front, step 5c. Unfortunately, the corner of the pant's pocket is not properly folded. To fix this, the bottom edge is picked up, moved toward the top of the page until the pocket corner is pulled tight, moved back down toward the bottom of the page, and then placed back down where it was at the start, step 5d. The final step is to top stitch the pocket edge at the sewing station

3.5 Discussion of Pants Assembly Line

The ultimate goal of automated manufacturing of apparel is 100% automation. Cloth or thread comes in one side of the plant and pants, shirts, and dresses go out the other side of the plant. No human hands would touch the threads of the garment. With the current technology this complete automation is simply not possible. It is not just a matter

of modifying the existing technology. Current $(TC)^2$ technology has certain fundamental limitations. However, it is important to realized just how much automation can be done with this technology.

The pants assembly line in Figures 3-1a-c is about 50% automated. It is not possible to say exactly how much is automated without defining what is meant by the percentage. The amount of automation could be judged by a number of different measures like the number of operations performed automatically, the length or number of seams sewn automatically, or the time of automated sewing. Probably the best measure is time. Take the time required to do the automated steps by hand, divide by the total time to make the garment, and multiply by one hundred. This will give the percentage of automation. Unfortunately the time required to perform each operation on the garment is not easily determined. The exact definition is not very important. The important feature is that the pants are about half finished before they are touched by the seamstresses at the end of the assembly line. As mentioned earlier, there are a number of ways to get closer to fully automated pants manufacturing. In the long run, however, there will need to be another break-through in technology before 100% automation will be within our reach.

The big break-through that is needed is some form of three dimensional sewing. Three dimensional sewing is needed to attach sleeves onto shirts, waistbands onto pants, and bodices onto skirts to make dresses. These operations are nearly impossible using existing $(TC)^2$ technology. Various individuals or organizations have conceived of various ways of doing three-dimensional sewing, but none have been demonstrated. Most people believe that even if a method of three-dimensional sewing is developed, it will be slower and more costly than two dimensional $(TC)^2$ sewing. Hence, two dimensional sewing will have its place in the future wherever it is applicable.

The Japanese have invested a lot of money into studying automated sewing systems. They performed worldwide patent searches and gave out several grants to investigate the possibility of developing an automated apparel factory. One of these factories is shown in Figure 3-3 and is called Automated Factory of "Upper Wear."[6] A modified version of this factory also appeared in an article in "Bobbin" magazine, but this time it was called Conceptional Ideas of Automated Sewing System, see Figure 3-4. It is important to notice that this is a conceptional idea of a factory and not

AUTOMATED FACTORY OF "UPPER WEAR"

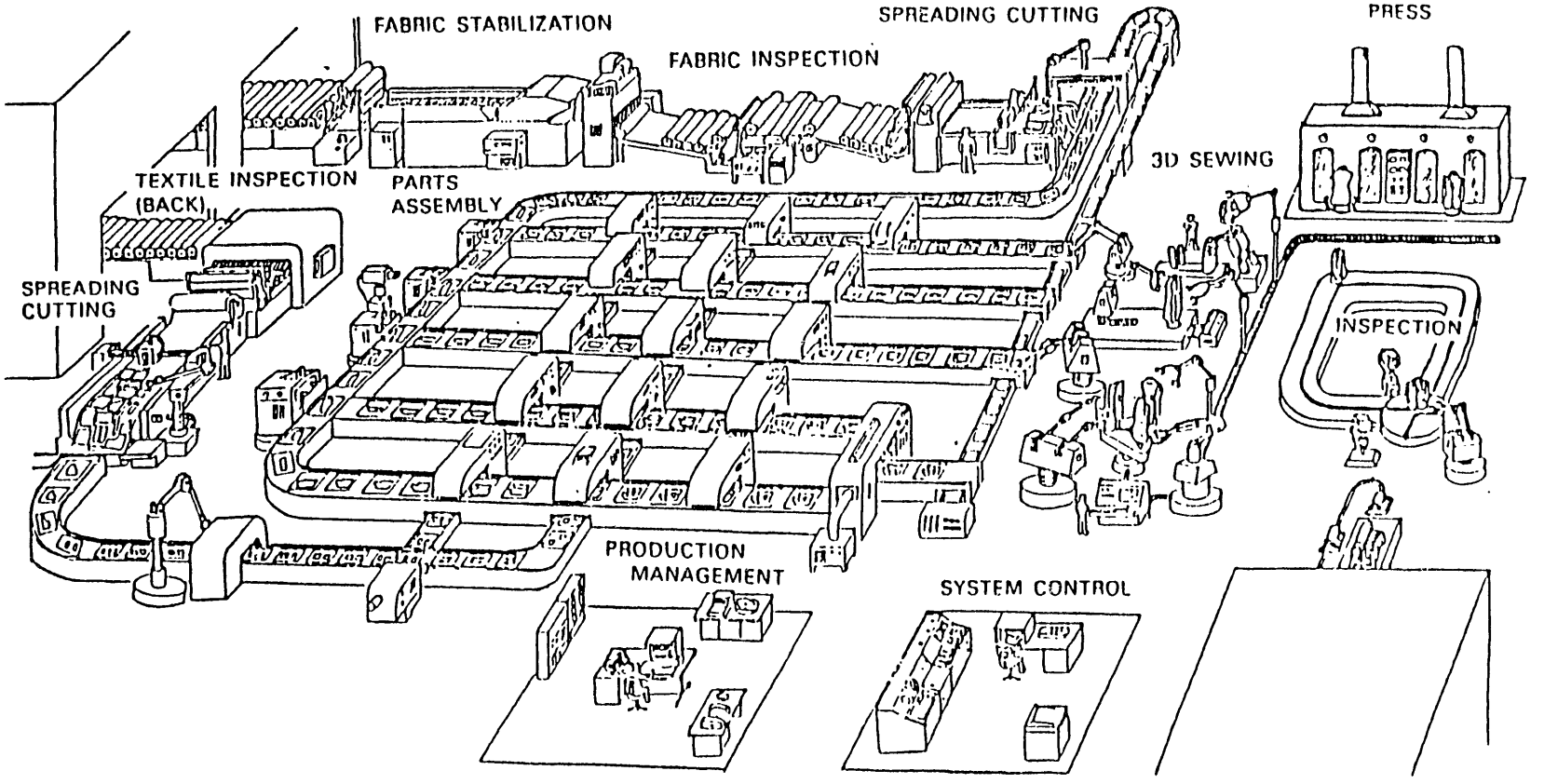


Figure 3-3: Automated Factory of "Upper Wear".

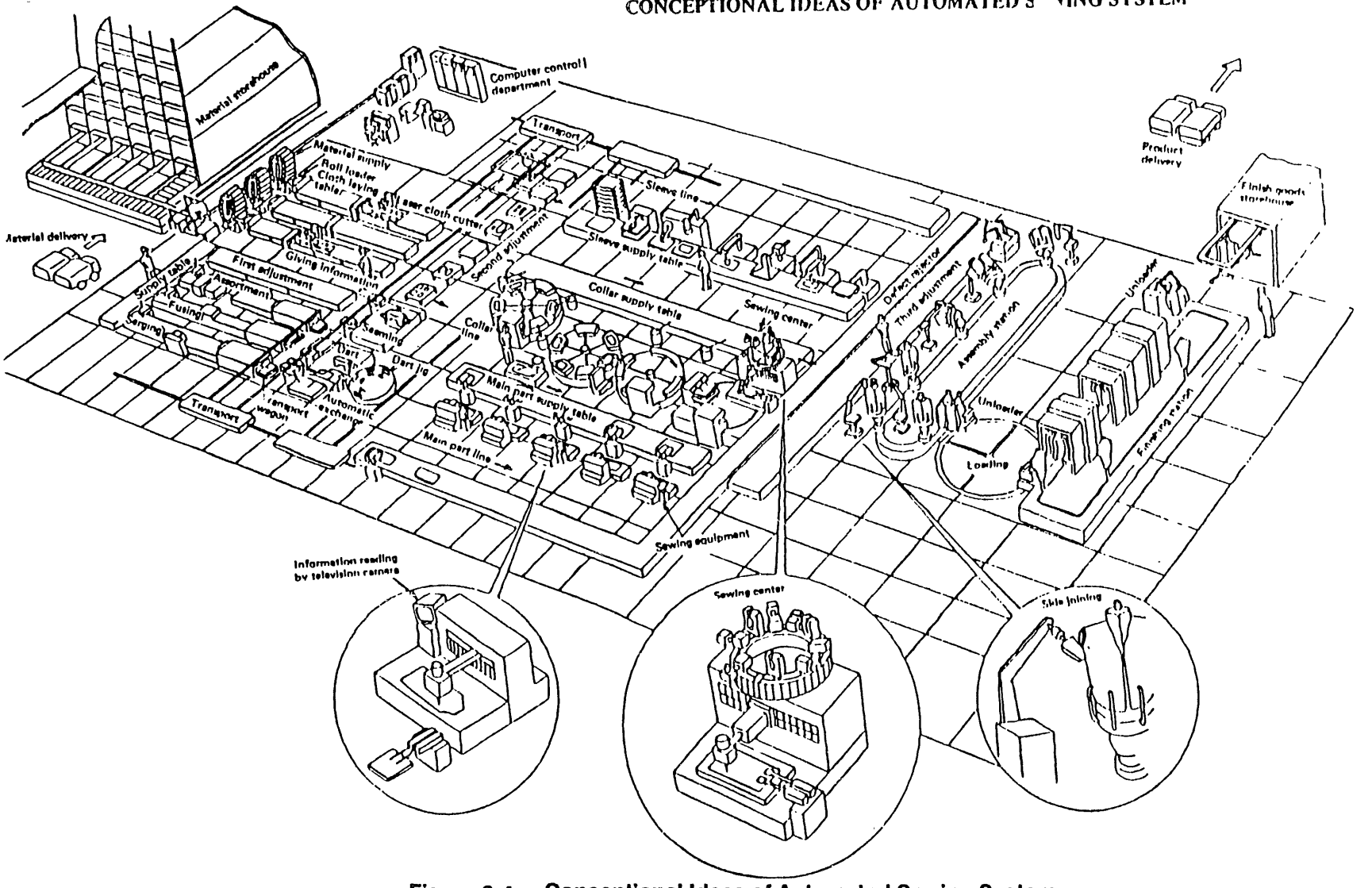


Figure 3-4: Conceptional Ideas of Automated Sewing System.

something that the Japanese are prepared to build. While much of these factories are involved with two dimensional sewing, both have some form of three dimensional sewing. In one factory that section is labeled *3D Sewing* and the other factory it simply says *Assembly station*. In either case they have a robot with one or two arms putting cloth pieces on a mannequin, sewing the seams, and using a vision camera for guidance. This is probably not impossible, but it will be a long while before we see a robot sewing a sleeve onto a shirt while the shirt is on a mannequin. So far a robot is not even capable of putting a sleeve on a mannequin's arm. These statements of lack of current ability are not meant to discourage anyone from attempting 3D sewing, they are only intended to clearly point out how far the current technology is from 3D sewing. More basic research, such as this thesis, is needed before super-apparel-factories are possible.

4 Non-linear Cloth Beam Bending Model

4.1 Introduction

One of the frequent failures that occurs during automated folding operations on the $(TC)^2$ machines is that one of the edges of cloth will get folded under after a pick-and-place operation. This occurs because too large an area of cloth is left unsupported during the transport process. In other words, the distance between the pickers and the edge of the cloth is too large. The apparent solution is to move the picker closer to the edge of the cloth, but this is not usually practical. Most types of pickers cannot function very close to the cloth edge. Another issue is that the robot has a limited number of pickers, and hence cannot support the whole edge of the piece of cloth. Picker locations must be chosen carefully and the best location for pickers is not generally obvious.

Understanding cloth deflections is one of the keys to understanding optimum picker placement. If, hypothetically, a suit sleeve were made of steel instead of cloth, the deflections of the sleeve could be calculated relatively easily. Just input the sleeve geometry and boundary conditions (picker locations) into a finite element analysis (FEM) program, like PATRAN, and in a few minutes the

computer would display the answer. But this solution does not work for cloth. Cloth typically exhibits large deflections that make the linear approximations in most FEM program invalid. This author knows of no large deflection plate model FEM program that will work for the cloth pick up problem, although some may exist. There is no doubt that such a program would be of value. However, simpler approximations to the large deflection plate model could be as useful, and in some cases more useful, in understanding the behavior of the cloth. These simpler approximations provide the necessary insight into optimum picker placement. In this thesis, simple approximation to cloth behavior will be used.

4.2 Bernoulli-Euler Beam Model

One simplified cloth model has received some attention in the literature [7,8]. This is the large deflection version of the Bernoulli-Euler beam model, also known as the elastica problem. The standard Bernoulli-Euler beam model has a number of assumptions that are applicable to many typical engineering design problems [9]. Most of these assumptions are also good for cloth - except one. Standard Bernoulli-Euler beam theory assumes that deflections are small. When deflections are not small, usually the case for cloth, the problem becomes substantially more difficult to solve.

Bernoulli-Euler beam theory states that bending moment is proportional to radius of curvature

$$M = E \cdot I / r \qquad (4\text{-}1)$$

where M is the bending moment in lbf/in (N/m), E·I is the bending stiffness in lbf·in^2 (N·m^2), and r is the radius of curvature in inches (meters), see Figure 4-1. Mathematically, radius of curvature is related to the first and second derivative of y with respect to x, hence equation (4-1) can be rewritten as

$$\frac{M}{E \cdot I} = \frac{\ddot{y}}{\{1 + \dot{y}^2\}^{3/2}} \qquad (4\text{-}2)$$

Where $\ddot{y}$ is d^2y/dx^2 and $\dot{y}$ is dy/dx. If dy/dx is small then equation (4-2) can be rewritten as

$$\frac{M}{E \cdot I} = \ddot{y} \qquad (4\text{-}3)$$

This is the standard engineering form of the Bernoulli-Euler beam equation used for most structural engineering problems. If dy/dx is small, then $(dy/dx)^2$ is small compared to one, and one raised to the 3/2 power is still one, hence the right hand side of (4-2) goes to d^2y/dx^2. However, if dy/dx is not small, then equation (4-2) must be used instead of (4-3). Equation (4-2) can be rewritten as

$$\frac{d\theta}{ds} = \frac{M}{E \cdot I} \qquad (4\text{-}4)$$

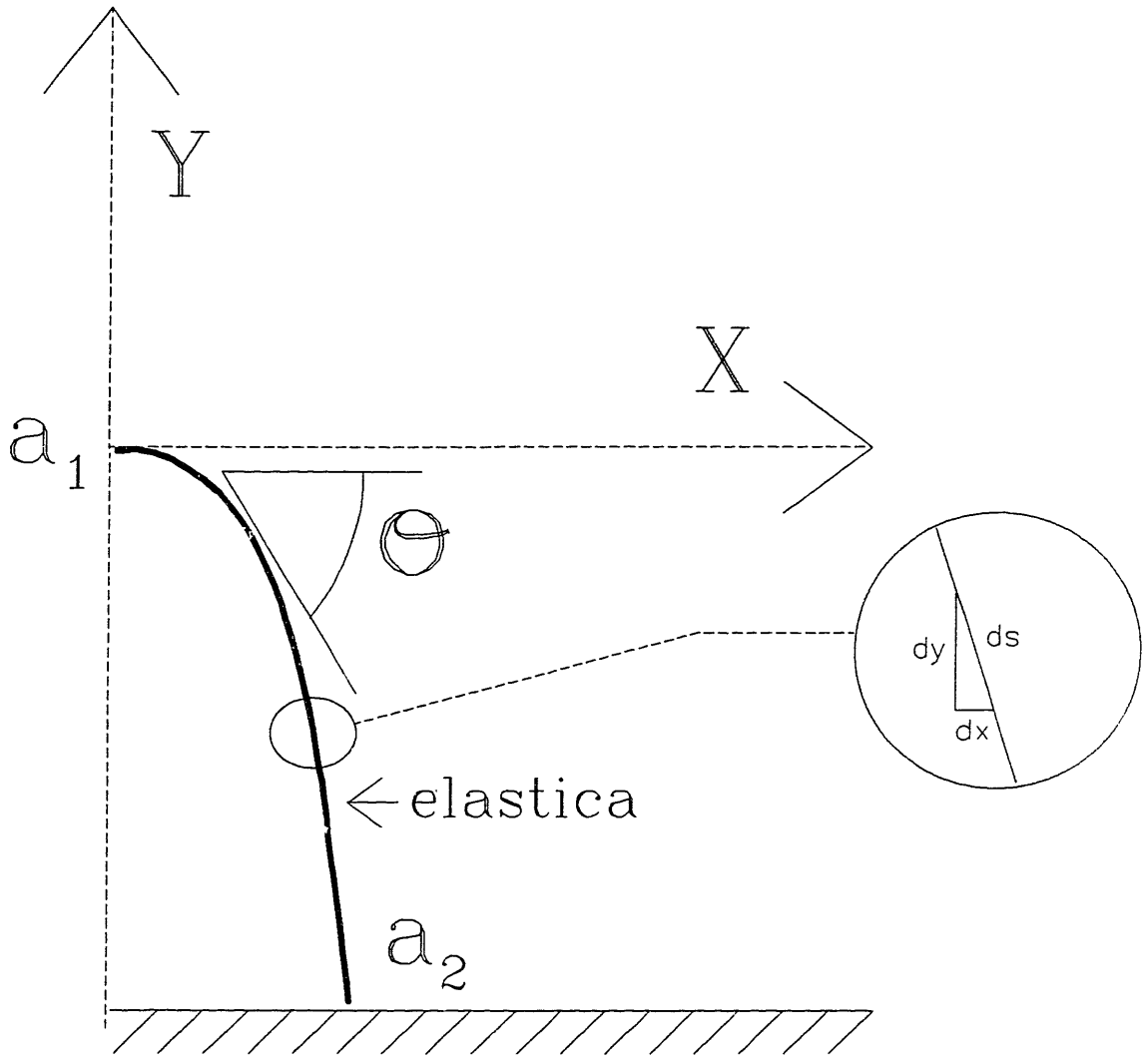


Figure 4-1: Elastica geometry.

Where θ is the angle between a tangent to a beam segment and the horizontal and s is the arc length.

Along with Equation (4-4), a set of equilibrium equations are needed to solve the elastica problem. Figure 4-2 shows the force/moment equilibrium requirements for a beam segment. These equilibrium requirements give rise to three differential equations.

$$dF_x/ds = 0 \qquad (4\text{-}5)$$

$$dF_y/ds = w \qquad (4\text{-}6)$$

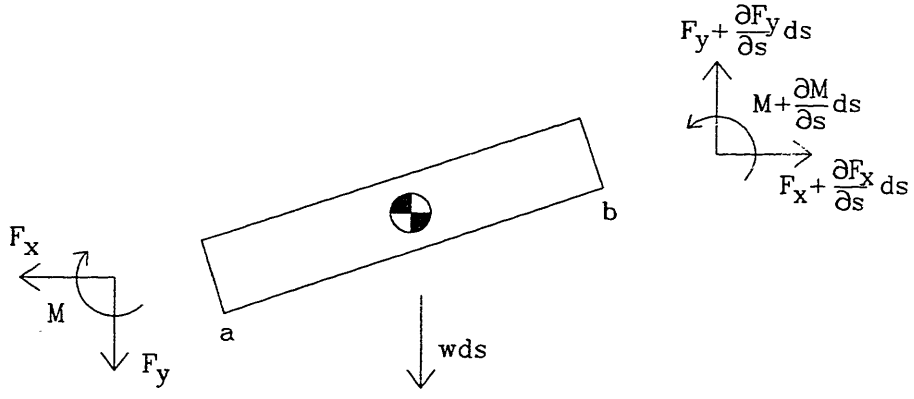
$$dM/ds = - F_y \cdot \cos(\theta) + F_x \cdot \sin(\theta) \qquad (4\text{-}7)$$

Here w is the weight per unit length of the beam (elastica). Equations (4-4) through (4-7) along with appropriate boundary conditions define the complete elastica problem.

When working on heavy elastica problems it is customary to use a specific dimensionless parameter, β. This parameter simply relates the elastica stiffness to its weight. It is dimensionless and is defined by

$$\beta^{1/3} = \frac{L}{(E \cdot I/w)^{1/3}} \qquad (4\text{-}8)$$

Where L is the unsupported length of the elastica and E•I and w have been defined previously.



F_x,F_y	Forces in the x and y directions
M	Moment about the Z axis
w	Weight per unit length of beam
a,b	Points at the ends of the element
ds	differential segment length

For Equilibrium

$$\sum F_x = 0 = -F_x + F_x + \frac{\partial F_x}{\partial s} ds$$

$$\boxed{\frac{\partial F_x}{\partial s} = 0}$$

$$\sum F_y = 0 = -F_y + F_y + \frac{\partial F_y}{\partial s} ds - wds$$

$$\boxed{\frac{\partial F_y}{\partial s} = w}$$

$$\sum M_b = 0 = -M + M + \frac{\partial M}{\partial s} ds + \cancelto{0}{(wds)(\tfrac{1}{2}dx)} + F_y dx - F_x dy$$

$$\frac{\partial M}{\partial s} = -F_y \frac{dx}{ds} + F_x \frac{dy}{ds}$$

$$\boxed{\frac{\partial M}{\partial s} = -F_y \cos\Theta + F_x \sin\Theta}$$

Figure 4-2: Static Equilibrium of Beam Element.

4.3 Computer Solution of Elastica Problem

Equations (4-4) through (4-7) have no known exact solution, hence they must be solved numerically. This set of four first order differential equations can be reduced to one single non-linear differential equation. (Note that the non-linearity is due to geometry and not due to non-linear behavior of the material.) Reducing the four equations down to one does not improve insight into the nature of these equations, hence it will not be done here. Instead, a numerical method will be used to solve the set of first order equations. Such equations can be solved by many different methods, such as finite difference, finite element, or integration methods. The method used here is the shooting method using a fourth-order Runge-Kutta integration.

Of the great number of integration methods available Runge-Kutta has an edge over other techniques. Runge-Kutta achieves fourth order accuracy with fewer calculations than many other methods. Furthermore, "Runge-Kutta succeeds virtually always: but it is not usually the fastest."[10] It is very easy to code. A discussion of benefits of the Runge-Kutta method can be found in "Numerical Recipes" by W. H. Press. The Runge-Kutta method will not be derived here,

the reader is referred to any elementary book on numerical methods. Appendix B shows the form of the Runge-Kutta integration used to solve equations 4-4 through 4-7.

The Runge-Kutta method can be used directly to solve initial value problems (IVP) for the heavy elastica. That is, if the values of F_x, F_y, M, and θ are given at one end of the elastica, we can integrate along the length of the elastica to calculate the forces and deflections. Frequently, however, not all the boundary conditions are known at one end of the beam. Instead, some boundary conditions are known at one end and some are known at the other. This is known as a two point boundary value problem, which is a more difficult to solve than the initial value problem. Techniques such as relaxation methods and shooting methods can be used to solve boundary value problems. The latter was used in this work.

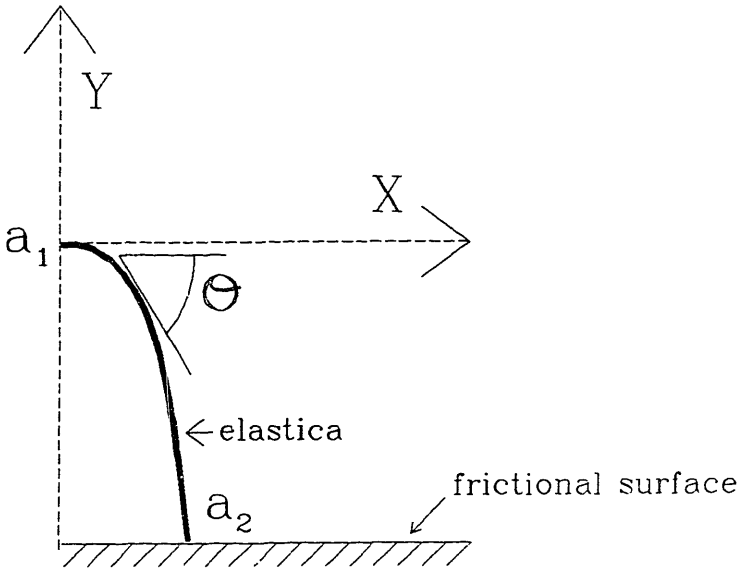
4.4 Shooting Method Solution of Boundary Value Problem

The shooting method provides a way to use simple integration techniques for two point boundary value problems. As already mentioned in the IVP, the solution can start at the beginning and march along using numerical integration to the end. For the boundary value problem the boundary conditions at one end are incomplete and

integrating from these incomplete boundary conditions is almost certain not to satisfy the boundary conditions at the end point. To overcome this problem, the shooting method employs a multi-dimension Newton-Raphson root finding technique. As an analogy, consider firing a cannon at an enemy camp. You must set the incline of the cannon (boundary condition at the beginning) so that its cannon ball will hit the enemy camp (boundary condition at the other end). To do this, a trial shoot is fired and based on where it lands the incline of the cannon is corrected. The shooting method works in the same way.

At the beginning boundary point, call this a_1, some but not all of the conditions are specified, see Figure 4-3. The boundary conditions at a_1 that are not known are placed in a vector $\vec{V}$. At the other end boundary point, call this point a_2, again some boundary conditions are known and some are not. In order to integrate the set of ODE's, the boundary conditions in $\vec{V}$ are guessed. By integrating the set of ODE's from a_1 to a_2 using the guessed $\vec{V}$, a trial set of boundary conditions is found for a_2. Since some of the boundary conditions were known ahead of time at a_2, there will be a discrepancy between the trial set of boundary conditions and the known boundary conditions. The

Unknown boundary conditions at a_1
$\vec{V} = \begin{bmatrix} F_{x0} \\ F_{y0} \\ M_0 \end{bmatrix}$



Known boundary conditions at a_2		
Case 1 just before contact	Case 2 contacting/sticking	Case 3 contacting/slipping
$\vec{F} = \begin{bmatrix} F_{xn} \\ F_{yn} \\ M_n \end{bmatrix}$	$\vec{F} = \begin{bmatrix} X_n \\ Y_n \\ M_n \end{bmatrix}$	$\vec{F} = \begin{bmatrix} F_{xn} \\ Y_n \\ M_n \end{bmatrix}$

Notation		
x	=	Cartesian Coordinate
y	=	Cartesian Coordinate
Θ	=	Polar angle
Fx	=	Force in x direction
Fy	=	Force in y direction
M	=	Moment about Z axis
0,n	=	subscripts, node numbers

Figure 4-3: Boundary conditions for BEAM.EXE program.

difference between these two sets of boundary conditions is placed in a discrepancy vector $\vec{D}$. The goal of the shooting method is to eventually find the value of $\vec{V}$ that zeros $\vec{D}$.

Multidimensional Newton-Raphson method is used to find the value of $\vec{V}$ that zeros $\vec{D}$. One by one each value in $\vec{V}$ is varied by a small amount ΔV, and a separate integration of the 4 ODE's is performed followed by an evaluation of

$$\alpha_{ij} = \frac{\partial D_i}{\partial V_j} \approx \frac{D_i(V_1,\ldots V_j+\Delta V_j,\ldots) - D_i(V_1,\ldots V_j,\ldots)}{\Delta V_j} \qquad (4\text{-}9)$$

Once the matrix $[\alpha]$ is determined a corrected guess of $\vec{V}$ can be made using the following equations:

$$[\alpha]\cdot\delta\vec{V} = -\vec{D} \qquad (4\text{-}10)$$

$$\vec{V}^{new} = \vec{V}^{old} + \delta\vec{V} \qquad (4\text{-}11)$$

In order to calculate the new guess of $\vec{V}$, i.e. $\vec{V}^{new}$, equations (4-10) and (4-11) show that it is necessary to invert $[\alpha]$. Once $\vec{V}^{new}$ is calculated, the set of ODE's can be integrated again, and this time $\vec{D}$ should be smaller. If the equations are linear, then $\vec{D}$ will be zero. If the equations are not linear, then the shooting method must be repeated until $\vec{D}$ is suitably small. Note that each round of the shooting method requires 4 integrations of the ODE's: one to find the current degree of mismatch, and one each for

the three variations of $\vec{V}$. This shows why boundary value problems are considerably more difficult to solve than IVP's.

For the problem of a cantilevered elastica contacting a friction surface, there are three different cases of the boundary value problem. These three cases are: 1) just prior to contact; 2) contacting and sticking; and 3) contacting and slipping. For each of these three cases; the conditions on the built-in end (point a_1) of the cantilevered elastica do not change, hence the vector $\vec{V}$ does not change either,

$$\vec{V} = \begin{bmatrix} F_{x0} \\ F_{y0} \\ M_0 \end{bmatrix} \qquad (4\text{-}12)$$

where F_{x0}, F_{y0}, and M_0 are the forces and the moments at the built-in end. At the other end of the elastica, point a_2, the boundary conditions change depending on which contact case is encountered. Thus the discrepancy vector $\vec{D}$ differs for each case. For case 1, point a_2 is not contacting, and thus F_{xn}, F_{yn}, and M_n must all be zero. In case 2, the elastica is sticking at point a_2, hence x_n and y_n are specified. Also M_n is again zero. Finally for case 3, y_n and F_{xn} are known. The value of F_{xn} is equal to $\mu \cdot F_{yn}$, where μ is the static coefficient of friction. Again for case 3, M_n is zero. Therefore, in solving the given

elastica problem, the condition of the elastica, at point a_2, must be checked before each iteration to determine which case is in effect.

By "animating" the elastica on the computer, the complete picture of a heavy elastica contacting a frictional surface can be seen. The heavy elastica is started in "free space," i.e. no horizontal surface. Once the solution converges, a flat horizontal surface was placed, mathematically, at y_n (i.e. point a_2). Then point a_1 is moved down a small step, e.g. 0.002 inch (0.05 mm), in the y direction so that point a_2 is contacting the table. After the move, it is decided whether the elastica is sticking or slipping and the solution for that case is found (section 4.5). Point a_1 is then moved down another step, it is decided again whether the elastica is sticking or slipping at point a_2, and the solution is obtained accordingly. The cycle is continued until point a_1 is just a little bit above the table.

4.5 Hints on Solving Heavy Elastica Problem

Although the solution to the heavy elastica involves direct application of the shooting method, there are some "tricks" that facilitate the solving of the problem. First, it should be noted that the heavy elastica problem has more

than one solution - even the non-contact problem has three or more solutions as shown Figure 4-4. This being the case, the initial guess of $\vec{V}$ has to be close to the desired solution. When the elastica is not in contact with the frictional surface, a good guess of $\vec{V}$ is not too difficult. In this case, F_{x0} is zero and F_{y0} is equal to the total weight of the elastica, and the only difficulty is to find a good guess for M_0. For very stiff elasticas with very little deflection, the bending moment M_0 is close to the value for linear Bernoulli-Euler beam, i.e. $M_0(\max) = w \cdot l^2/2$ where w = load per unit longitudinal length and l = length of beam. For limp elasticas with large drape, however, the value of M_0 drops to about 25% of the $M_0(\max)$. Furthermore, if M_0 is set equal to $M_0(\max)$ for limp heavy elasticas, the solution will not converge. To overcome this problem a program was written to calculate values of M_0 vs. β (the heavy elastica parameter). The data from this program was then curve fit using a ninth-order polynomial. For a given value of β, an accurate guess of M_0 could be made using the resultant curve. The guess is usually so good that the solution converges on the first try. Once $\vec{V}$ is found for the non-contact problem there is no problem with the contact problem because the old value of $\vec{V}$ is close to the next value of $\vec{V}$. The use of the polynomial curve fit simply adds in the convergence of the solution.

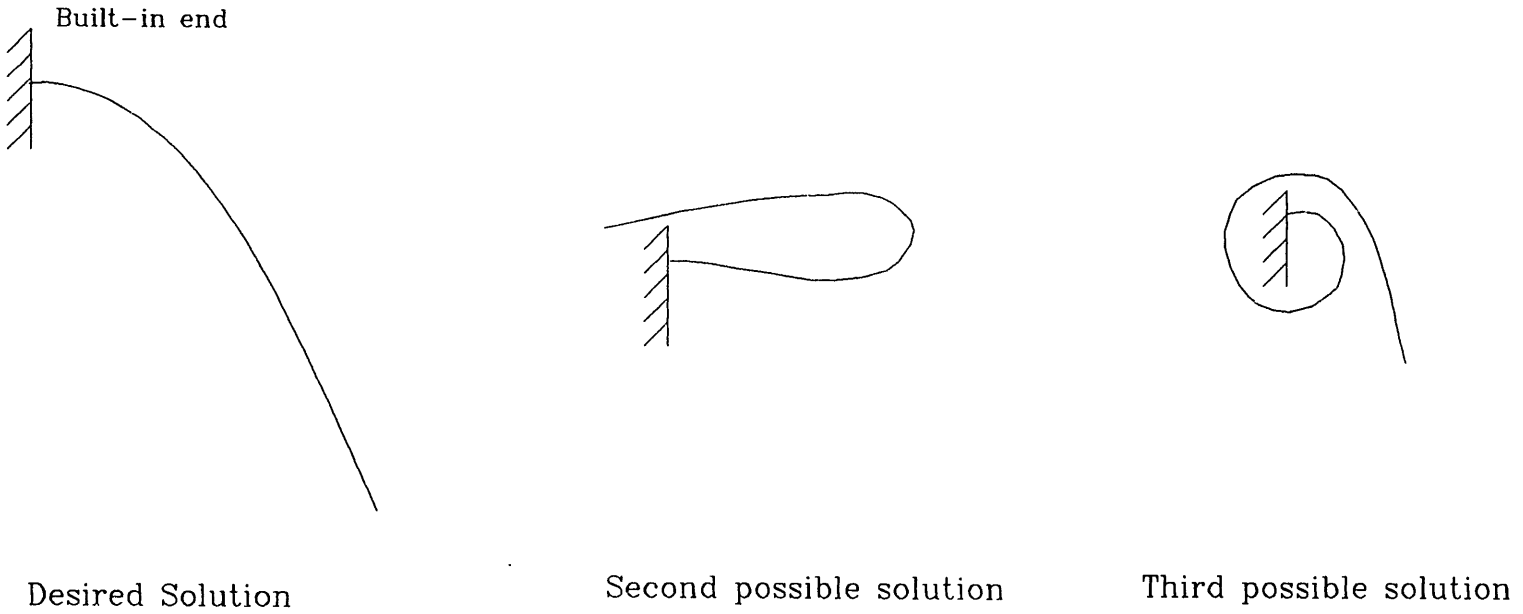


Figure 4-4: Multiple Solutions for Free Elastica.

In order to keep close to the solution at all times, $\Delta\vec{V}$ should be kept very small. This is very important, because otherwise the solution will not converge. The problem occurs because very small changes in M_0, F_{x0}, and F_{y0} cause large changes in the position of point a_2. This is particularly true of limp elasticas. It was found empirically that variation of M_0, $\Delta V[3]$, had to be less than or equal to $0.0000005 \cdot M_0$ in order for the iteration to converge for limp elasticas. If the variation of M_0 was large and the computer did not have double precision, the variation in $\vec{D}$ would be swamped by round off errors and the solution could not be found.

Another way to assure that the problem would not stray too far from the solution was by clipping $\Delta\vec{V}$. If the shooting method tries to change the values in the vector $\vec{V}$ by more than 10% at any step, then the change in $\vec{V}$ is limited to 10%. This prevents problems due to non-linearities in the equations. The shooting method assumes that the equations are sufficiently linear in the area where it is trying to find a solution. Frequently these linear approximations give the wrong numerical answer but the correct sign. Thus the clipping still allows the shooting method to get closer to the solution during each iteration.

When the shooting method is close to the solution, i.e within 10%, there is no needed to apply clipping and the solution can be found very quickly. The 10% clip limit stated here was just as an example and the actual program allow the user to set the clip limit. Clipping in no way affects the value of the final solution, it simply aids the program in finding the solution.

The criterion for deciding whether the beam is sticking or slipping must be chosen very carefully. Obviously the beam is slipping if the horizontal force, F_{xn}, is greater than the maximum frictional force, $\mu \cdot F_{yn}$. The solution is found when $F_{xn} = \mu \cdot F_{yn}$, however this could also be seen as the criterion for sticking. Sticking occurs when $F_{xn} < \mu \cdot F_{yn}$, so there could be some confusion when $F_{xn} = \mu \cdot F_{yn}$. It was discovered that the program would get stuck in a loop of sticking/slipping/sticking/slipping/etc. To avoid this, the criterion was changed to slip if $F_{xn} > 1.005 \cdot \mu \cdot F_{yn}$ and stick if $F_{xn} < 0.995 \cdot \mu \cdot F_{yn}$. Thus the program would not get stuck in a loop when near the solution. If $0.995 \cdot \mu \cdot F_{yn} < F_{xn} < 1.005 \cdot \mu \cdot F_{yn}$, then the solution will slip if it had been previously slipping, and stick if it had been sticking.

One final note of caution. The numerical solution of the frictional contact elastica problem is not very robust. In attempting to solve these heavy elastica problems one

must be very careful that errors do not build up. Specifically this can be a problem when determining if a solution has converged. It is common practice to terminate the iteration when $|\text{error}| < \epsilon$, where ϵ is a small number. For this problem it is frequently better to terminate the solution when $0 \leq \text{error} < \epsilon$.

4.6 Results of Heavy Elastica Program

The heavy elastica/frictional contact problem was solved for a range of coefficients of friction and a range of elastica lengths. Observing the solutions to these problems revealed three fundamentally different modes of behavior. The three modes are 1) complete slip, 2) stick then slip, and 3) complete stick. These three modes are illustrated in Figures 4-5, 4-6, and 4-7, respectively.

The *complete slip* mode occurs with low coefficients of friction and/or short elasticas. Before the elastica contacts the table, the tip angle at the free end is relatively small. Because of this, the elastica is relatively stiff in response to forces in the x direction. Therefore, when the elastica contacts the table it slips in response to frictional forces. As the elastica moves vertically downward a double reversed curvature is formed in the elastica, see Figure 4-5. The elastica ends up lying flat on the frictional surface.

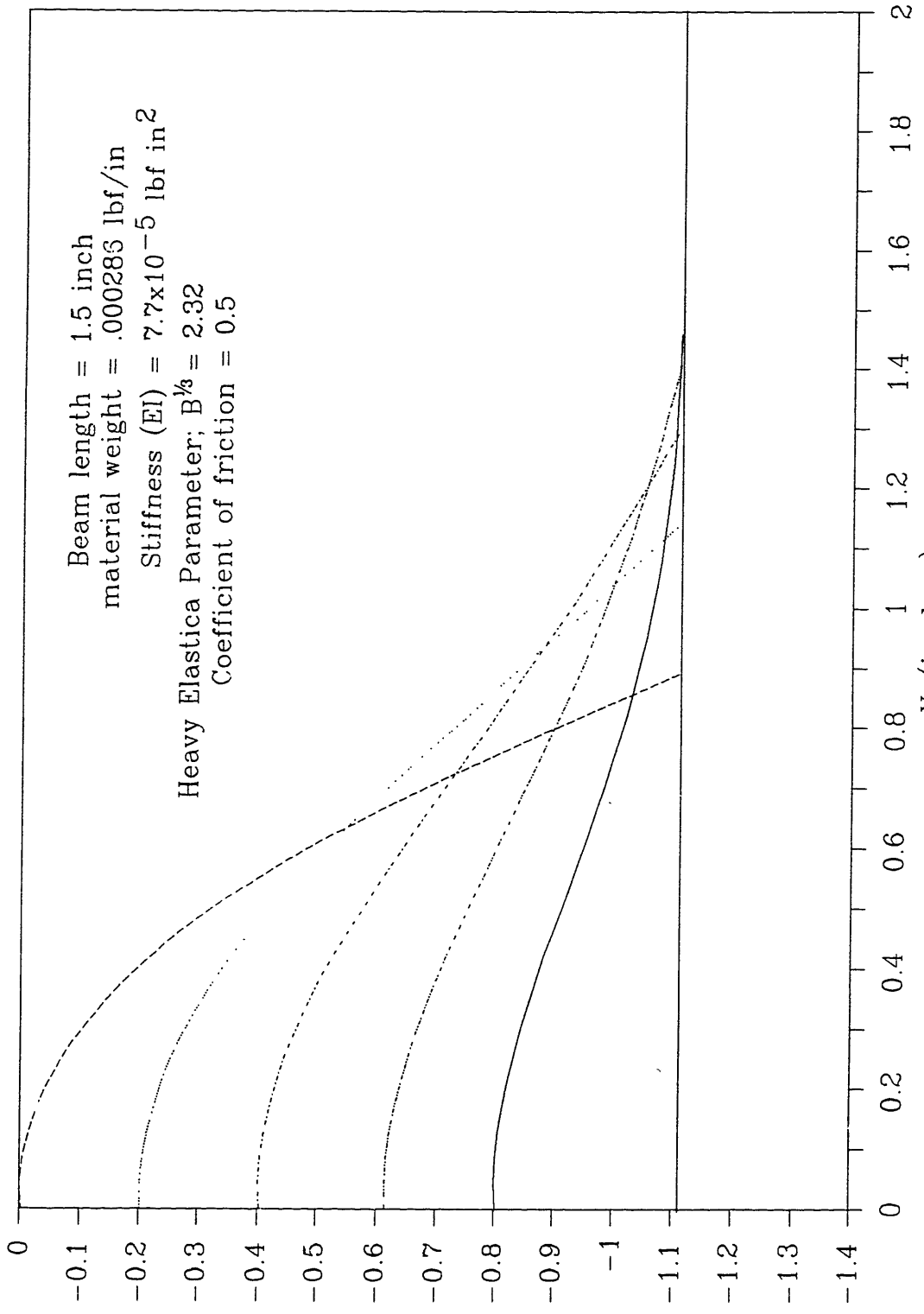


Figure 4-5: Theoretical Results for 1.5 inch long Elastica with Coefficient of Friction =0.5. Complete Slip Mode.

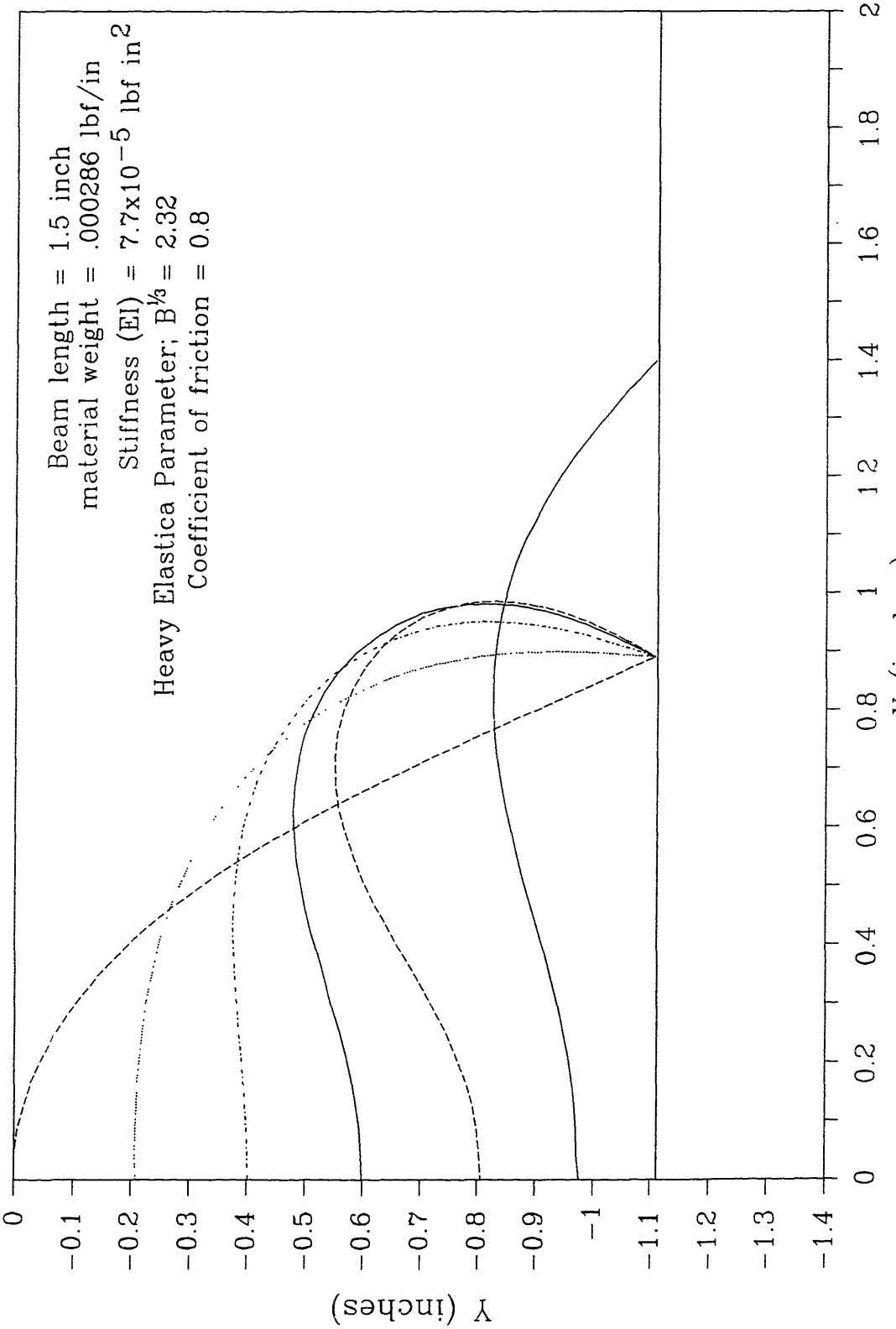


Figure 4–6: Theoretical Results for 1.5 inch long Elastica with Coefficient of Friction= 0.8. Stick then Slip Mode.

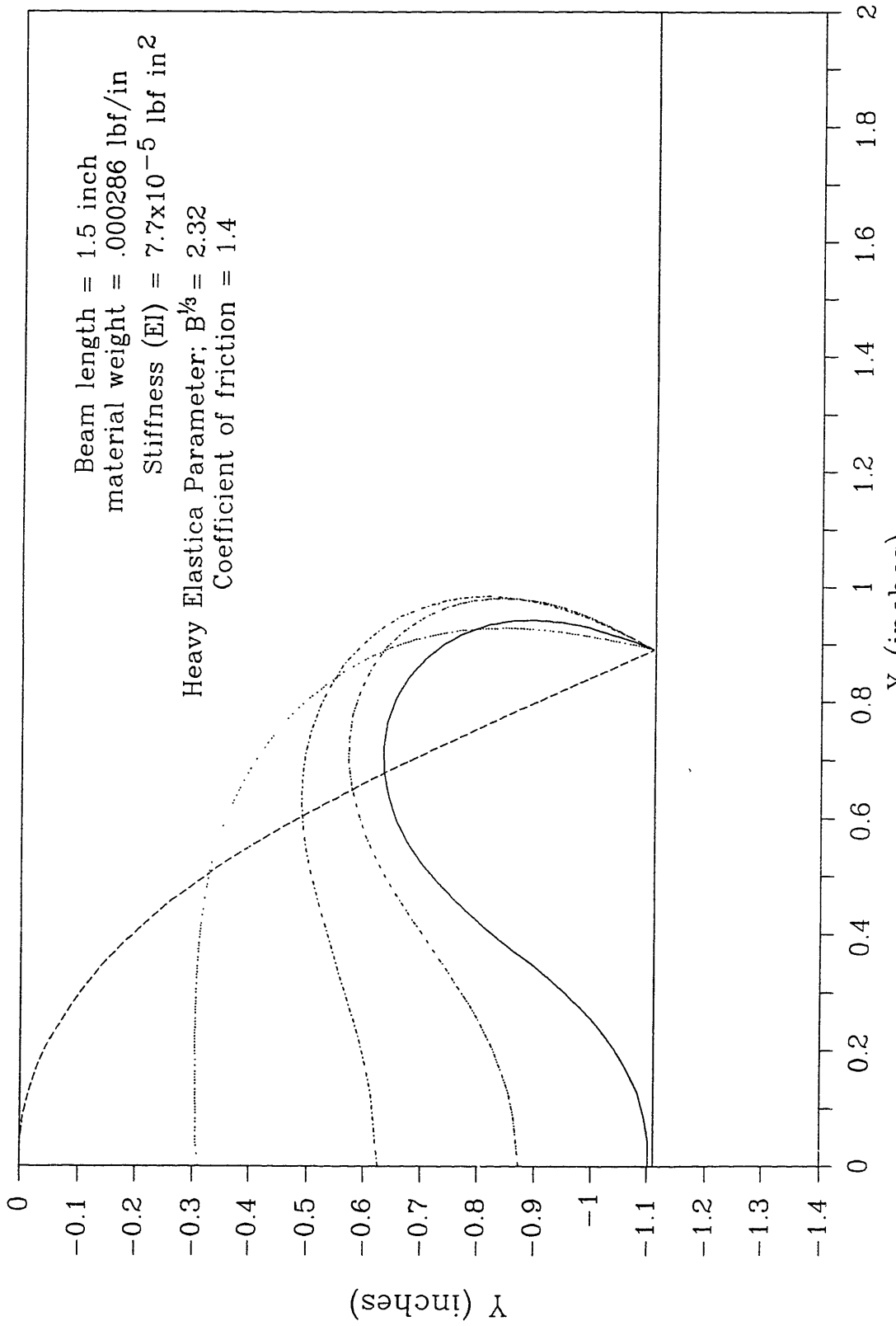


Figure 4-7: Theoretical Results for 1.5 inch long Elastica with Coefficient of Friction =1.4. Complete Slip Mode.

The *stick then slip* mode occurs at moderate values of friction and elastica lengths. As elastica length is increased, so does the vertical deflection of the elastica. Due to the increased moment arm, the effective horizontal stiffness of the elastica is reduced. This causes the free end of the elastica to stick in response to horizontal frictional forces. As the built-in end is moved vertically downward, the elastica is forced into a increasingly tighter radius of curvature. As the radius of curvature gets smaller, the horizontal force on the free end of the elastica increases. If the coefficient of friction is not too high, then the elastica will *flip out* resulting in the elastica lying flat, not folded, on the frictional surface as shown in Figure 4-6.

The complete stick mode occurs with high coefficients of friction and long elastica lengths as depicted in Figure 4-7. In this case, as the built-in end of the elastica is moved vertically downward, the free end is "held" in place by frictional forces. The elastica forms a loop that is approximately a semi-circle. Since the horizontal force at the free end never exceeds the "break away" force, the end point remains fixed. It should be noted that the free end of the elastica does not want to slip in the negative X

direction under any circumstances. An elastica in the complete stick mode ends up folded under itself on the frictional surface.

For elasticas with $\beta^{1/3} \geq 2.95$ the *stick then slip* mode does not occur. Under this condition, as the built-in end of these longer elasticas are moved down, the elastica is again forced into an increasingly tighter radius of curvature. But this time the force needed to maintain that curvature is supplied vertically upward by the frictional surface as illustrated in Figure 4-8 and 4-9.

As described above, the mode of behavior of the elastica depends only on the coefficient of friction and the heavy elastica parameter $\beta^{1/3}$. Figure 4-10 summarizes the conditions under which the three different modes occur. For values of μ and β in the lower left region of the graph, slipping occurs. For values of μ and β in the upper right region of the graph, sticking occurs. For values of μ and β in the remaining region, *stick then slip* occurs. As noted previously, for $\beta^{1/3} > 2.95$, the *stick then slip* mode does not occur. This mode graph represents a very compact way to depict what is expected to happen in a given situation. For example, if μ is given, and we want to know what is the longest elastica that will exhibit pure slipping. Simply

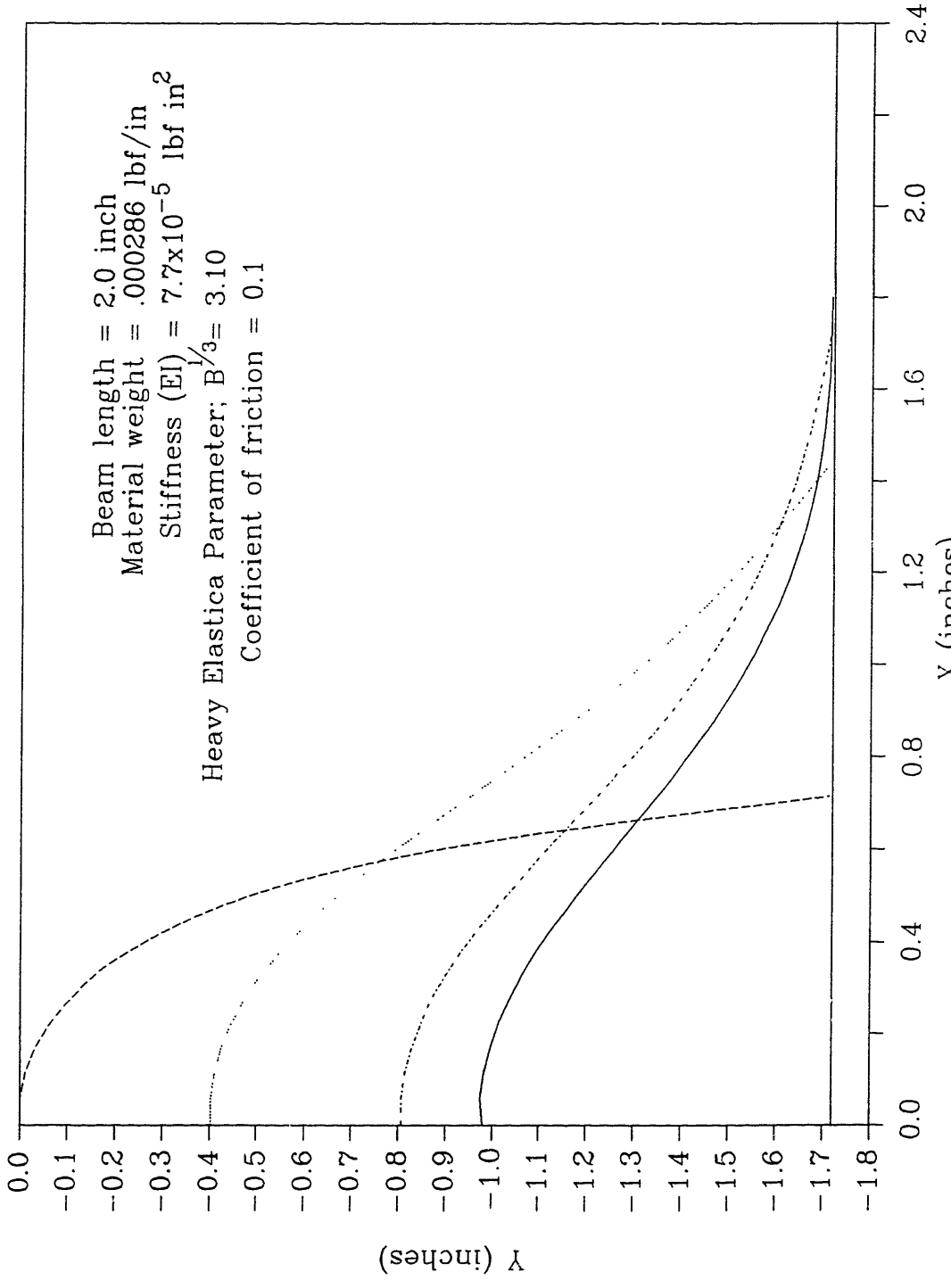


Figure 4-8: Theoretical Results for 2.0 inch long Elastica with Coefficient of Friction = 0.1. Complete Slip Mode

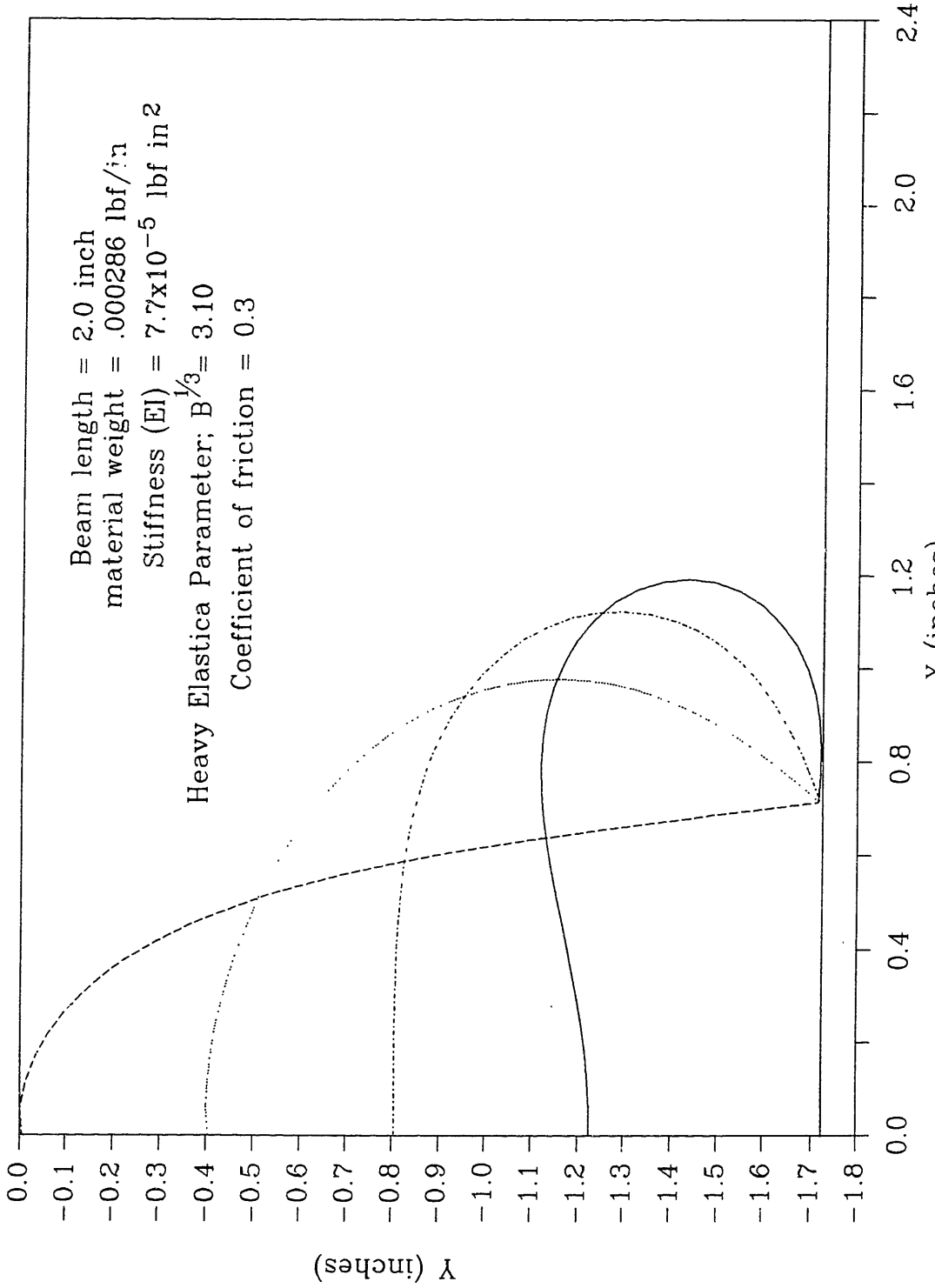


Figure 4-9: Theoretical Results for 2.0 inch long Elastica with Coefficient of Friction = 0.3. Complete Stick Mode

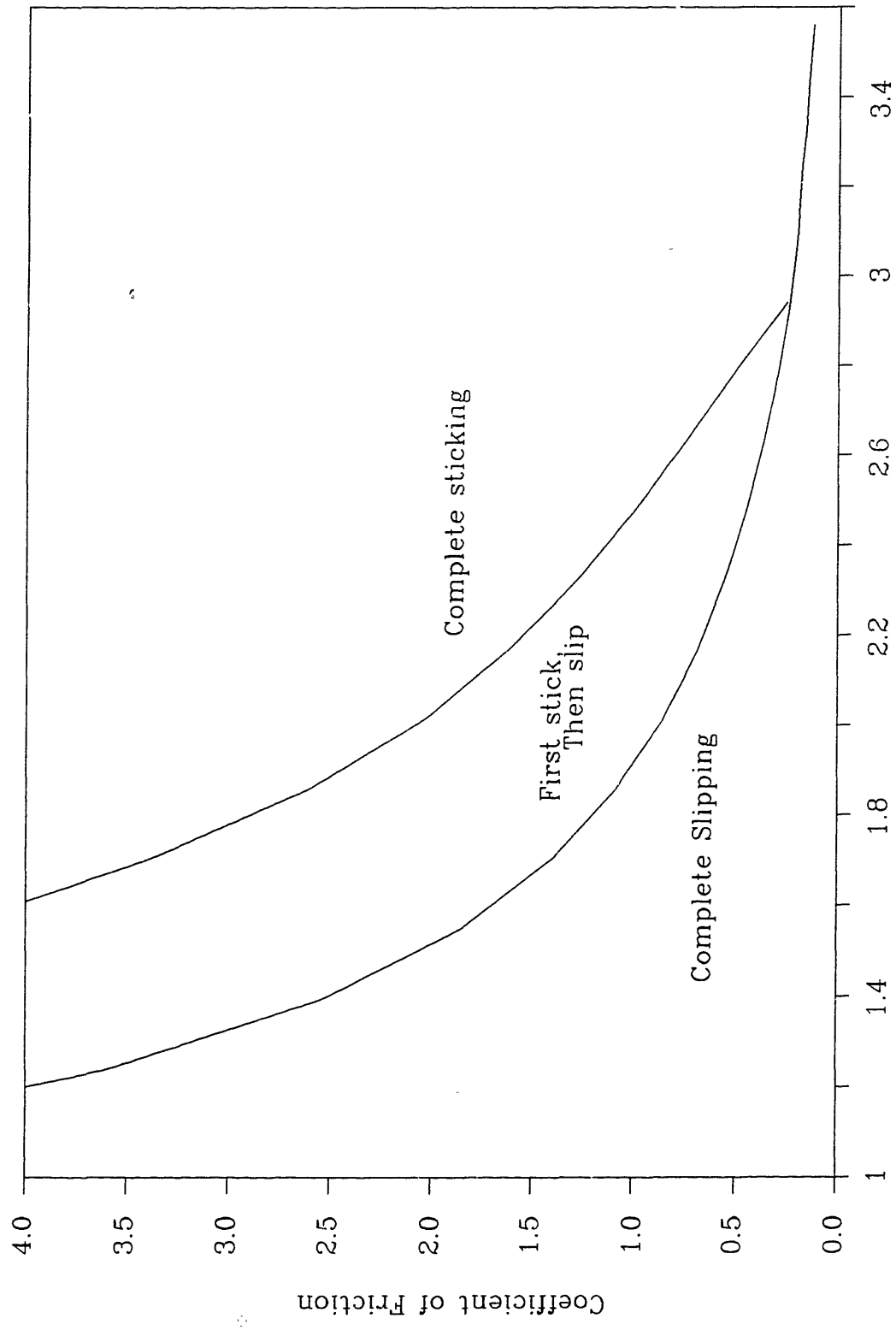


Figure 4-10: Heavy Elastica Mode Graph
Contact on Frictional Surface.

look up on the graph, follow μ over to the slip line and down to $\beta^{1/3}$. Knowing E·I and w; L can be calculated from $\beta^{1/3}$. The graph in Figure 4-10 was plotted for fairly broad ranges of μ and $\beta^{1/3}$. The range of $\beta^{1/3}$ was stopped at 3.6 because the corresponding value of μ was getting too low for typical materials. In the next section, the validity of the mode graph in Figure 4-10, which was obtained theoretically, will be experimentally verified.

4.7 Experimental Results of Cloth Heavy Elasticas

In an attempt to verify the heavy elastica/frictional surface mode graph, a series of tests were performed with textile fabrics. A plain woven fabric with a known bending stiffness was used along with a series of plates with various coefficients of friction. Tests were performed over a small range of μ due to the difficulties in finding high friction materials.

The simplest way to measure coefficient of friction is by the classical method of inclined plane. For the inclined plane, the coefficient of friction is equal to the tangent of the angle of incline, see Figure 4-11. The test consists of placing a cloth sample, 4 inch x 5 inch (10 cm x 12.5 cm), on the flat friction surface and then lifting one end of the surface until the cloth slips. The angle is

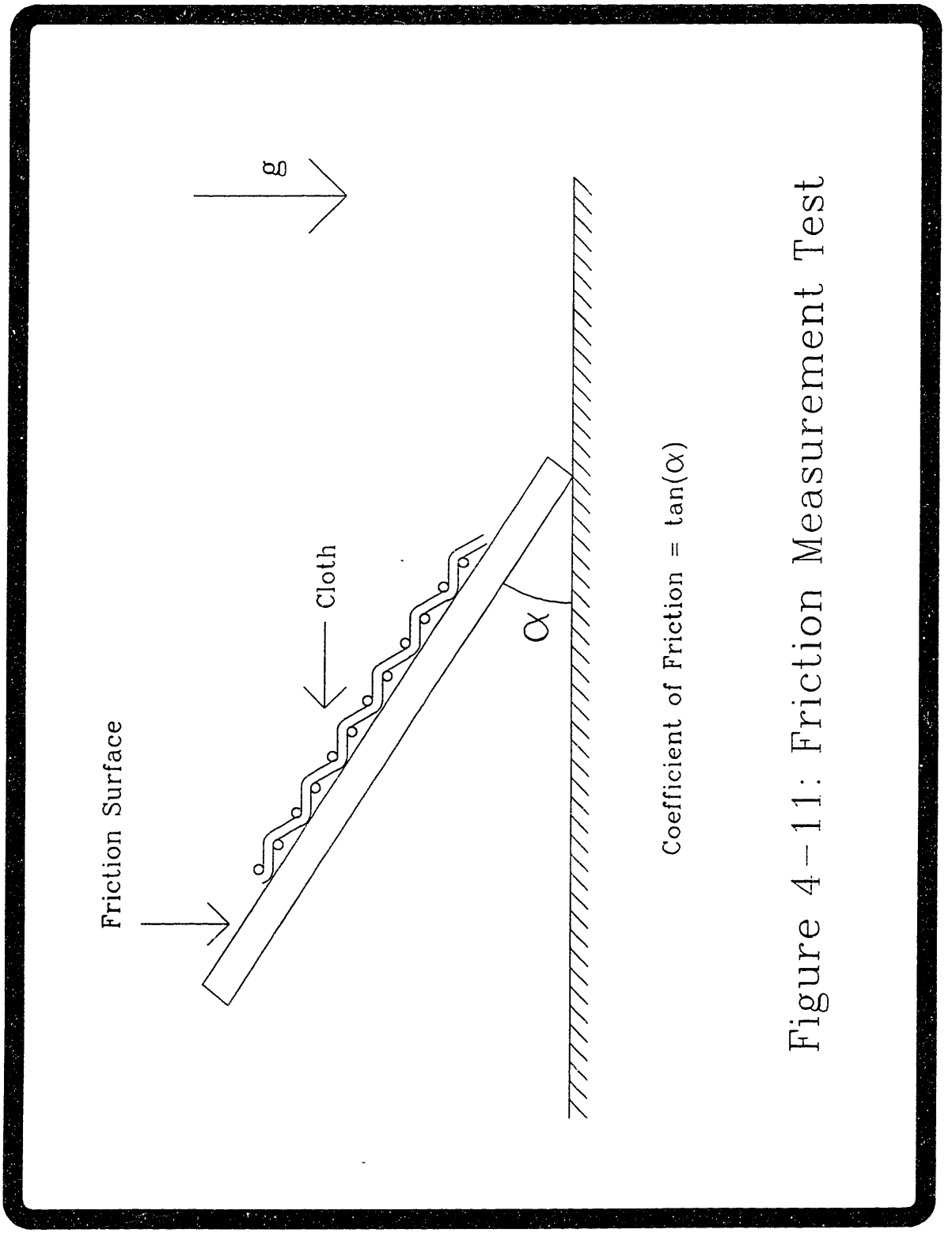


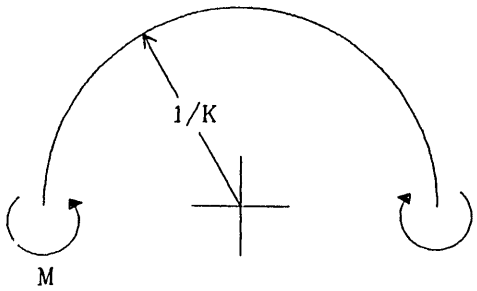
Figure 4-11: Friction Measurement Test

measured and the static coefficient of friction can be calculated. In measuring the coefficient of friction it is important to measure it under conditions that closely simulate the actual conditions. For the heavy elastica/frictional contact problem the end loads are very small. The inclined plane test uses only the weight of the cloth itself as the load. This is close to the actual load. However, the actual loading condition varies over a large range. When the free end of the elastica first contacts the frictional surface, the load is a small fraction of the total weight of the elastica. On the other hand, when the elastica is near its final configuration, most of the weight of the elastica is on the free end. It is reasonable to expect some change in coefficient of friction over such a wide range of pressures.

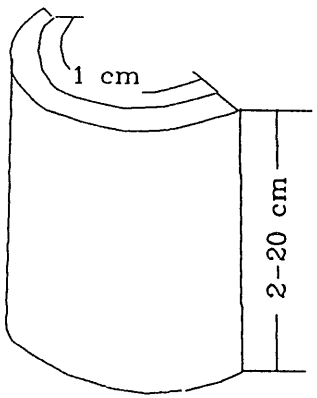
Two separate methods were used to measure the bending stiffness of the cloth. The first method was ASTM test D1388-64 *Standard Test Methods for Stiffness of Fabric*. The test consists of slowly pushing a rectangular piece of cloth off of a flat platform until a line joining the tip of the cloth and the edge of the platform makes a 41.5 degree angle with the horizontal. One half of the overhung length under this condition is called the *bending length*. The cube of the bending length times w (the weight per unit length of

the cloth) is the bending stiffness. The second method used to test bending stiffness was the Kawabata bending stiffness tester (courtesy of Prof. Timothy Clapp of North Carolina State University), see Figure 4-12. The Kawabata bending tester forces a sample of cloth (1 cm x 20 cm) through a range of radii of curvature and measures the torque required. The output of the bending tester is a graph of bending moment versus radius of curvature. The slope of this line is the bending stiffness. The units of bending stiffness per unit length are $gf \cdot cm^2/cm$, where gf is grams force. (Note that 1 kgf = 9.8 Newton. For an explanation see Marks' Standard Handbook for Mechanical Engineers, page 1-40. To convert to $lbf \cdot in^2$ multiply by $3.481 \cdot 10^{-4}$.)

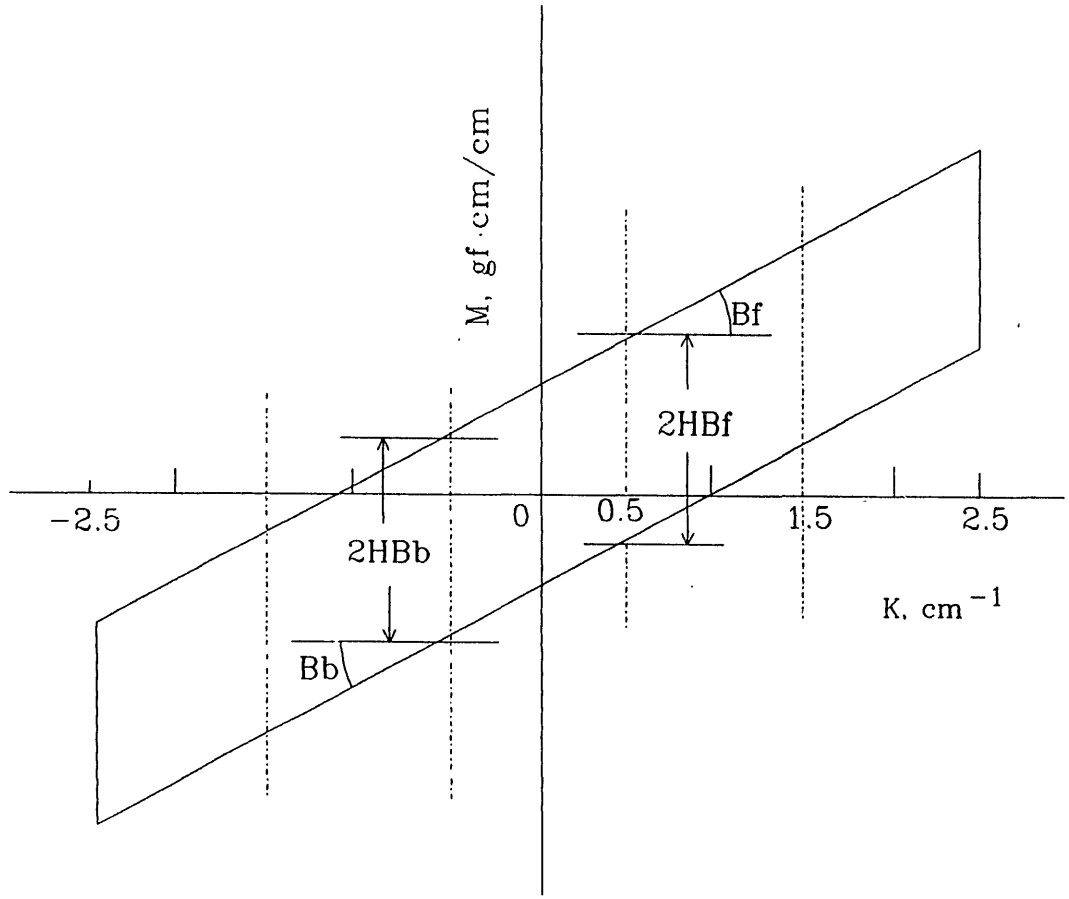
The results of the two methods for measuring cloth stiffness agree fairly well. The stiffness value from the Kawabata tester were found to be lower than the values obtained from the ASTM test. The values from the ASTM test were used for two reasons; 1) the ASTM test is closer to the actual physical situation being investigated; 2) the Kawabata tester was in North Carolina which made testing difficult. One important feature of the Kawabata tester is that it measures the amount of hysteresis in the material. A bending stiffness curve for a typical wool/poly suit



a) Plan view of bending geometry



b) 3-D view of bent cloth



c) Schematic of bending moment versus curvature graph

B; Bending rigidity per unit length (unit: gf·cm^2/cm)
2HB; Moment of hysteresis per unit length (unit: gf·cm/cm)

Figure 4-12: Kawabata bending stiffness tester

material is shown in Figure 4-13. For comparison, Figure 4-14 shows a bending curve for a typical piece of Xerox paper. Note the change of scale in Figure 4-14 and the signal saturation. The Xerox paper is about 20 times stiffer than the cloth. This shows just how limp a piece of cloth is. The hysteresis for the wool/poly suit material is not particularly large although it is probably not negligible. The elastica beam model does not account for hysteresis, hence there will be some difference between the model and the actual test results. The slope of the Kawabata curve is the bending stiffness. Notice in Figure 4-13 that the slope tends to fall off sightly for larger K (smaller radius of curvature). This too represents a problem for the model. The elastica model assumes that the bending stiffness is a constant. A better model would make the bending stiffness a function of K. For a heavy elastica, the radius of curvature has a wide range of values. The radius of curvature is small at the beginning of the elastica and large at the free end. Nevertheless, the value for the bending stiffness is only a weak function of K and using the average slope from the Kawabata curve is a reasonable approximation.

The heavy elastica mode graph is shown again in Figure 4-15 with some experimental data points. The experimental apparatus consisted of a camera enlargement mount that

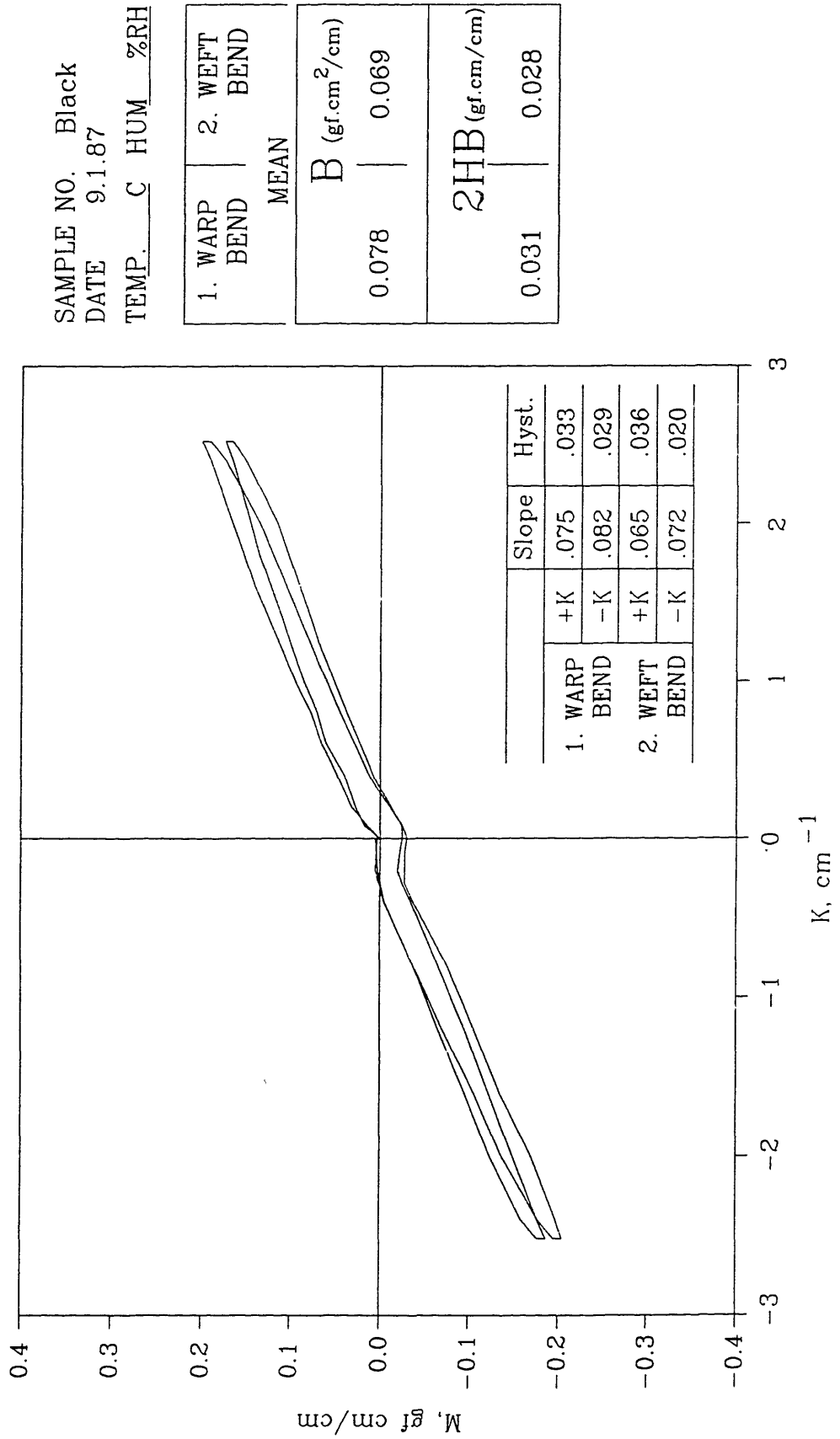


Figure 4−13: Kawabata bending stiffness test results for plain woven wool/poly suit material

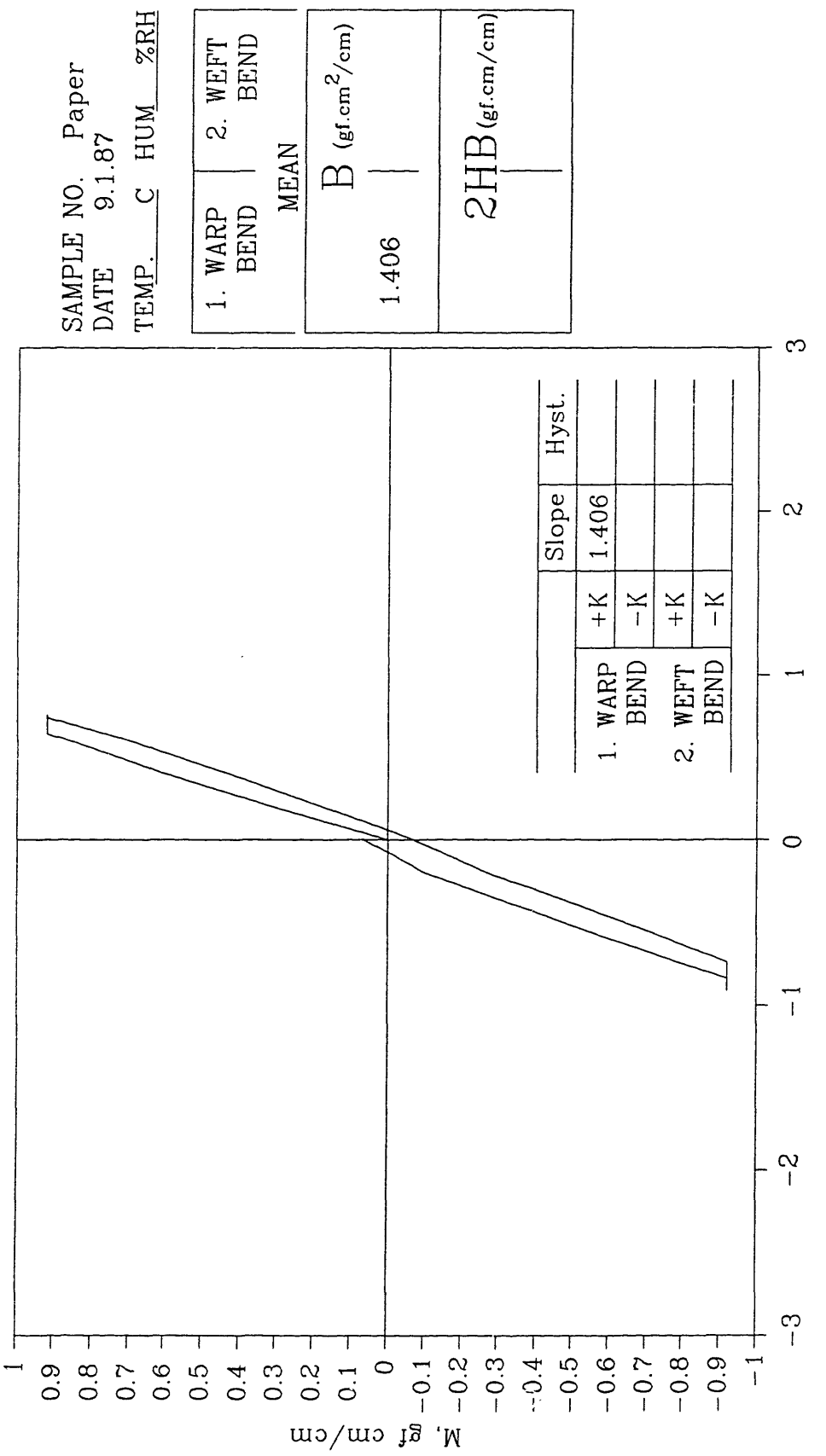


Figure 4-14: Kawabata bending stiffness test results for typical Xerox paper.

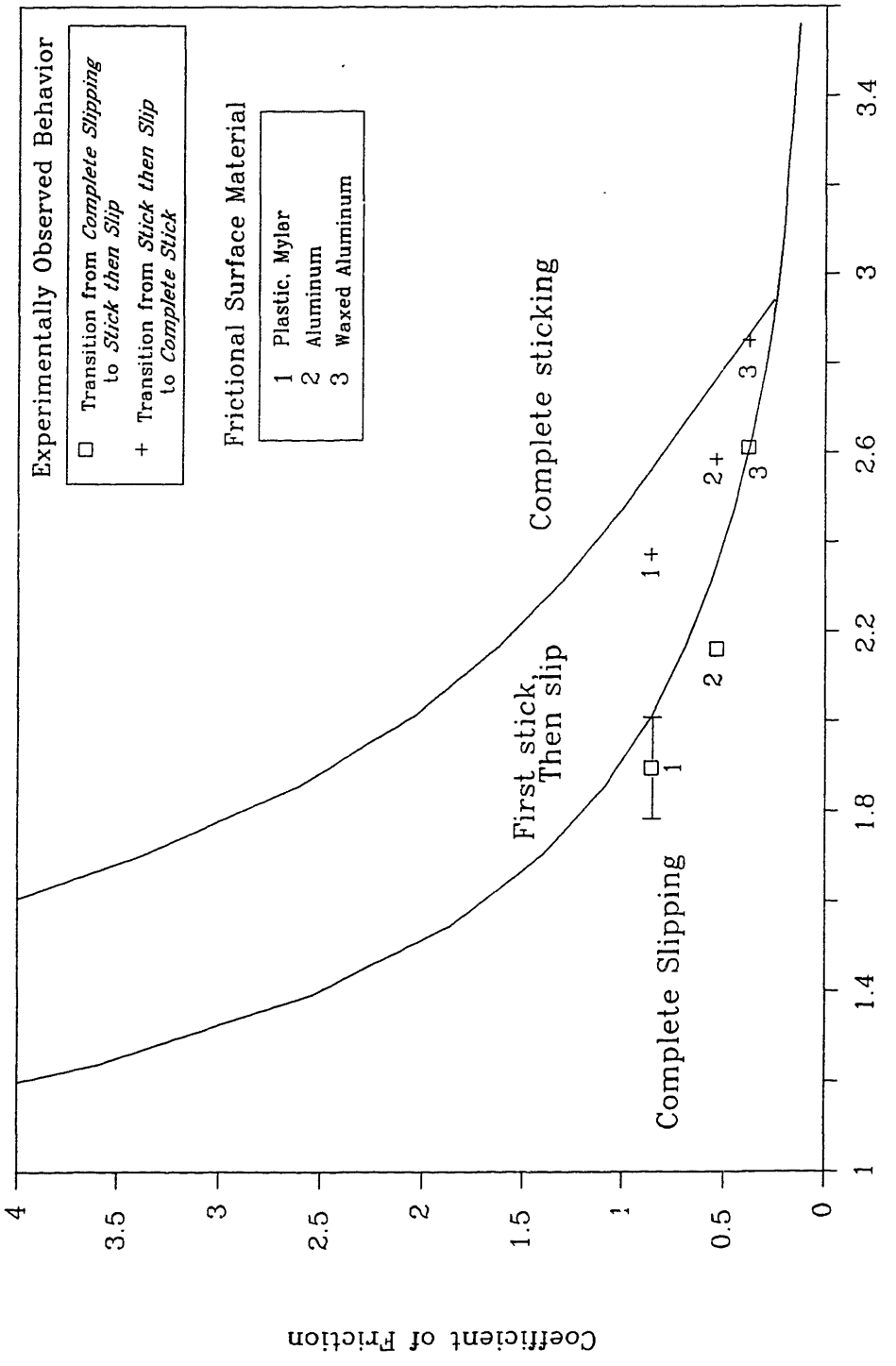


Figure 4-15: Heavy Elastica Mode Graph with Experimental Data Points.

could be moved up and down. The cloth was simply placed on the camera enlargement fixture with a fixed overhang and then slowly lowered onto a frictional surface. Three different frictional surfaces where used in the testing. Each combination of elastica length and frictional surface was perform four times to reduce statistical variations. The value at which mode transitions occur, were recorded. The experimentally observed behavior agrees with the theoretically predictions quite well. It can be seen that the shape of the experimental curve is about the same as the shape of the theoretical curves. In this comparison, the value of the bending stiffness has the largest uncertainty. The uncertainty in the frictional measurement is small enough that it is shown on the graph.

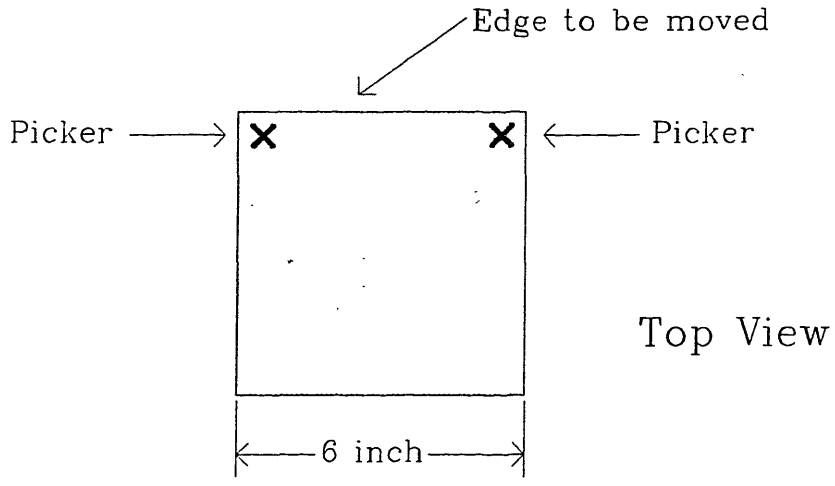
4.8 Conclusions

The behavior of a cantilevered piece of cloth contacting a frictional surface was investigated. An elastica beam model was developed and solved numerically. Experimental data was found to agree with this model. Three distinct modes of behavior were discovered: complete slip, stick-then-slip, and complete stick. The intended use for this model is as a *rule* that governs how close pickers must be placed to the edge of the cloth. Chapter 6 will demonstrate how the picker-to-edge rule will be used in determining picker locations.

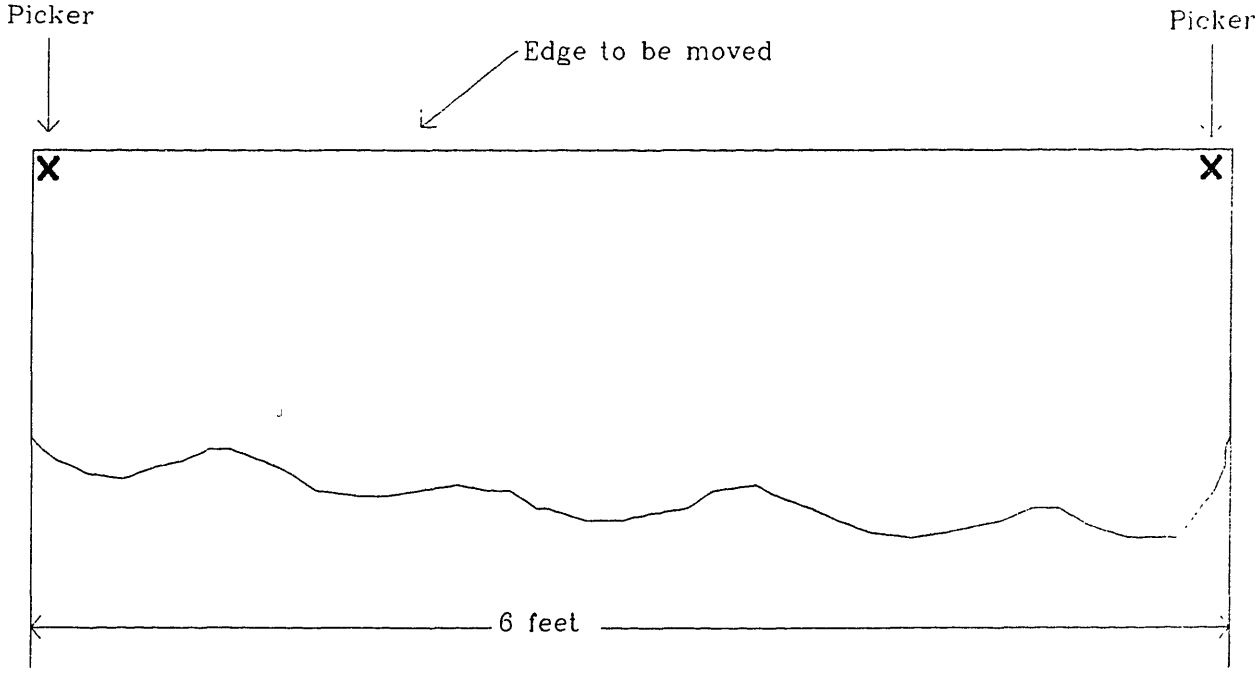
5 Rule for Inter-Picker Spacing

5.1 Inter-Picker Spacing Problem

In addition to the problem of how far pickers can be placed to the edge of the cloth (Chapter 4) is the problem of how close they should be placed to each other. To demonstrate the problem of the inter-picker spacing, consider trying to pick up the edge of a 6 inch (15 cm) square piece of cloth. The expert system must determine where to place the pickers. Based on the picker-to-edge criterion (i.e. the elastica model from Chapter 4) , pickers must be within the critical distance on each corner to avoid sticking, see Figure 5-1. Next, consider the same problem with a 6 foot (1.8 m) square piece of cloth. Using the picker-to-edge criterion, the pickers are placed on the corners just as before. But this time the cloth between the two pickers will not be adequately supported. If the edge of this 6 foot (1.8 m) square is moved to a new location and put down on the table, then the edge will almost certainly be distorted. The unsupported area of the cloth is simply too large. The cloth is hanging like a sail, and the position of the center point on the edge cannot be controlled.



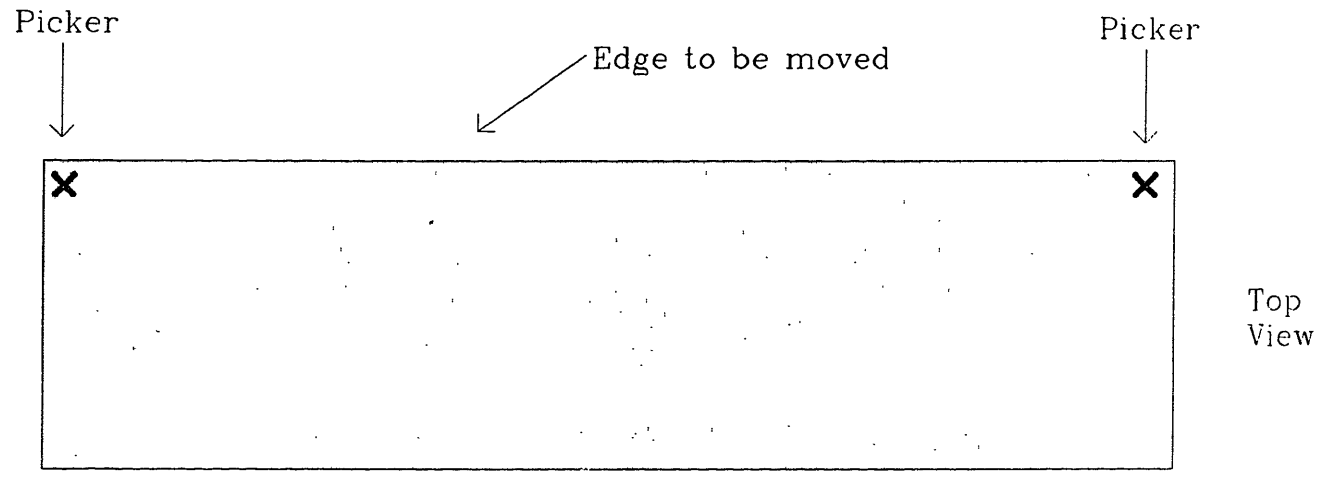
Six inch square being picked up by two pickers. Due to the short distance between the two pickers the edge being moved is unlikely to be distorted.



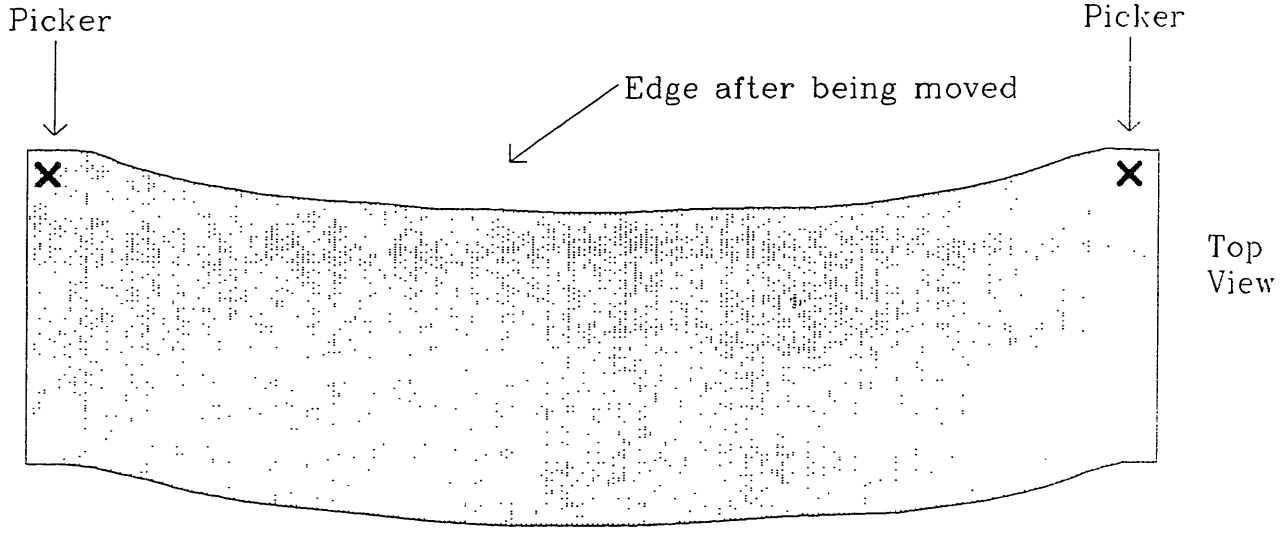
Part of a six foot square piece of cloth being picked up by two pickers. The long edge between the pickers is going to be moved. Due to lack of support the edge will get distorted as it is moved.

Figure 5-1: Picker Placement for Square Cloth Pieces

Two phenomena are responsible for the lack of control of long edges. On small pieces of cloth, the total shear stiffness between pickers is large enough that shear deflections between pickers are small. On large pieces of cloth with large inter-picker spacing, the cloth shear stiffness is low and the cloth can shear freely as shown in Figure 5-2. The second problem with large inter-picker spacing is large cloth deflections under the force of gravity as illustrated in Figure 5-3. The cloth between two pickers behaves like the cable of a suspension bridge and forms a shape similar to a catenary. With small inter-picker spacing, the catenary cannot form. If pickers are *sufficiently* close together, then the edge of the cloth will bend over like the heavy elasticas as discussed previously. With the cloth bent over like this the stiffness of the cloth between pickers is effectively increased, see Figure 5-4a. If the pickers are placed far enough apart the opposite happens. The cloth deflects between the pickers like a catenary. This catenary effectively stiffens the cloth between the pickers and the edge, see Figure 5-4b. For a moderate inter-picker spacing the cloth is bistable. It can deflect like a catenary or like a cantilever. Clearly for the 6 foot (1.8 m) square piece of cloth, more pickers are needed; and an inter-picker criterion must be developed for the expert system.

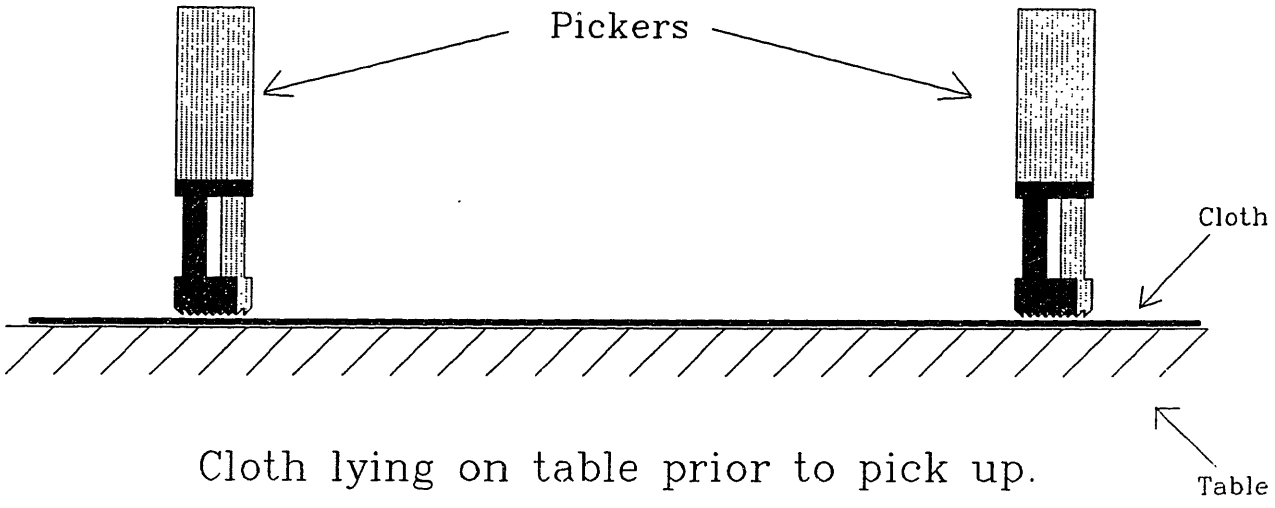


Rectangular piece of cloth prior to being moved. It will be slid toward the top of the page.

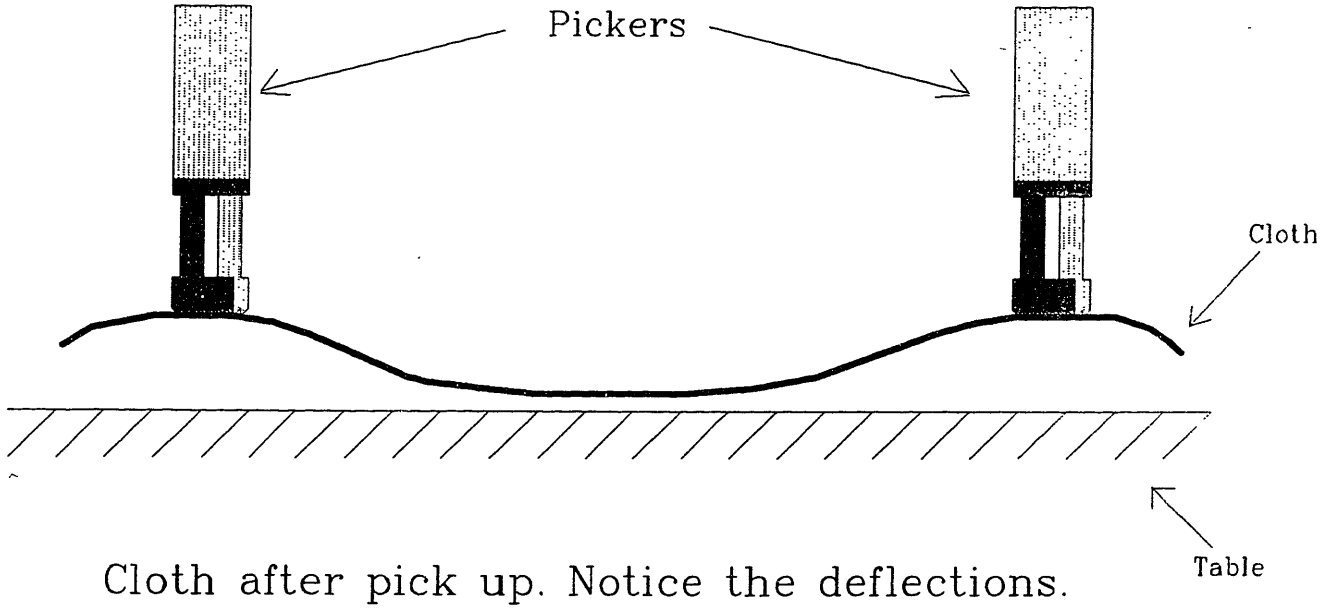


Rectangular piece of cloth after having been moved along the folding table. The exaggerated deflections illustrate the typical shearing deflections of the cloth. To aviod this problem the pickers would have to be placed closer together.

Figure 5-2: Excessive Shearing of Cloth Between Pickers



Cloth lying on table prior to pick up.



Cloth after pick up. Notice the deflections.

Figure 5-3: Cloth Catenary Deflection Between Pickers.

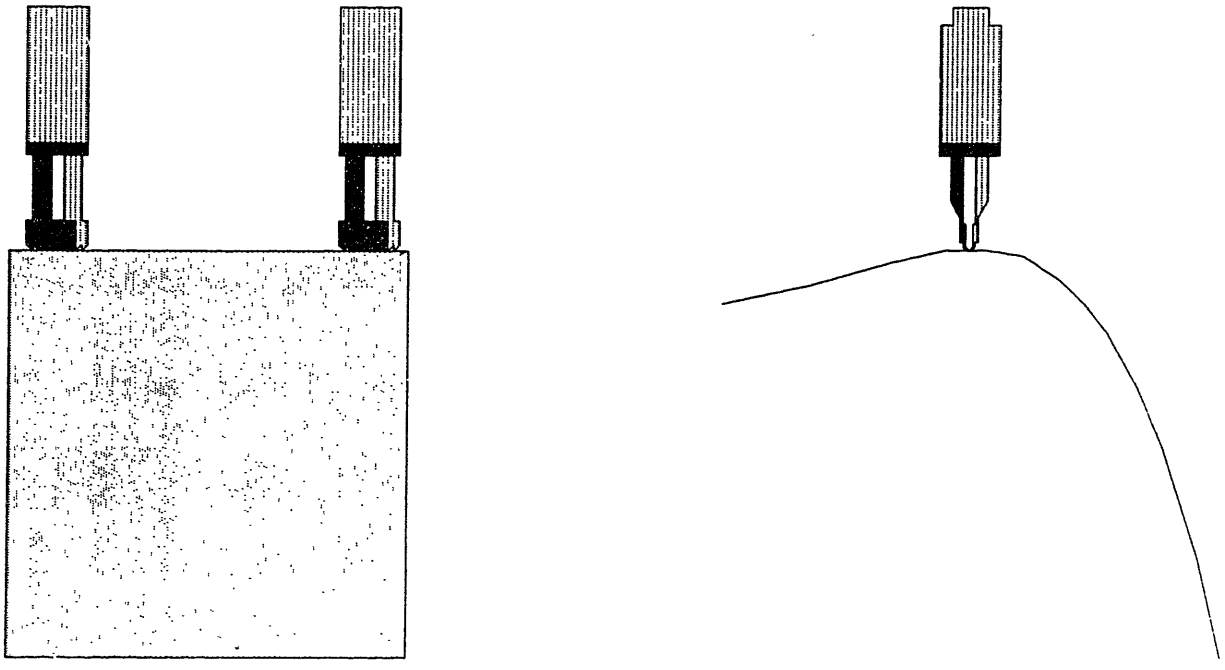


Figure 5-4a: The cloth between the two pickers bends like an elastica. This bending stiffens the cloth between the two pickers so that it does not form a catenary.

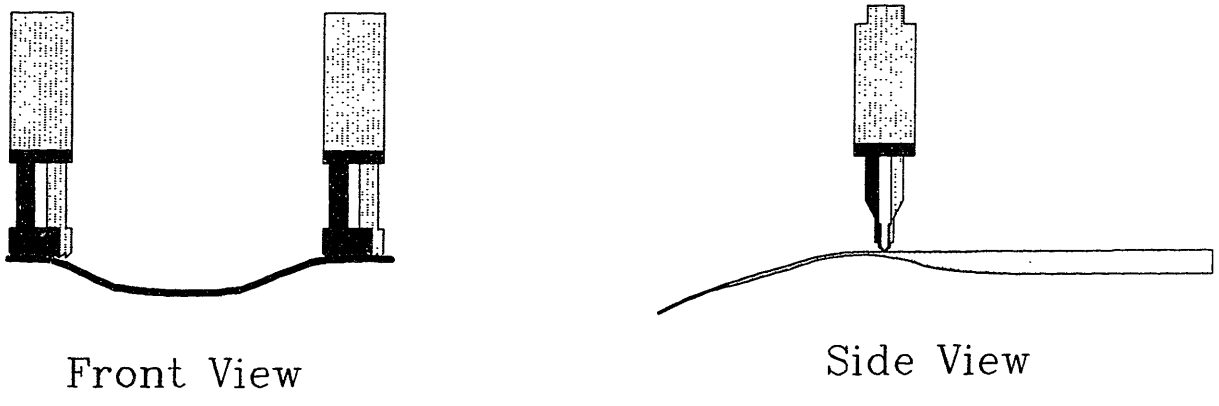


Figure 5-4b: The cloth between the two pickers bends like a catenary. This bending effectively stiffens the cloth so that the elastica does not form and the edge of the cloth sticks straight out.

5.2 Inter-Picker Spacing Criterion

The inter-picker spacing criterion is not always needed. To see this, consider trying to find a set of picker locations for a pants pocket, see Figure 5-5a. For typical suit material the picker-to-edge criterion (Chapter 4) would be about 1 inch (2.54 cm), and the inter-picker spacing criterion would be about 6 inches (15 cm). The first picker is placed at the right hand edge. The picker is placed within 1 inch (2.54 cm) of the edge to meet the picker-to-edge criterion. The placement of the next picker must meet both the picker-to-edge criterion and the inter-picker spacing criterion. First consider trying to satisfy the inter-picker spacing criterion. The second picker is placed 6 inches (15 cm) to the left of the first picker, see Figure 5-5b. This picker location satisfies the inter-picker spacing criterion but does not satisfy the picker-to-edge criterion. To see this, a line is draw between the two pickers. Then a line, parallel to this line, is draw tangent to the cloth edge. A rectangular area is drawn based on these two lines, see Figure 5-6a. This rectangular area is modeled as an elastica as discussed in the previous chapter. The width of this rectangle, and hence the length of the elastica, is 2 inches (5 cm). Therefore the picker-to-edge criterion is not met. Even if the picker on the

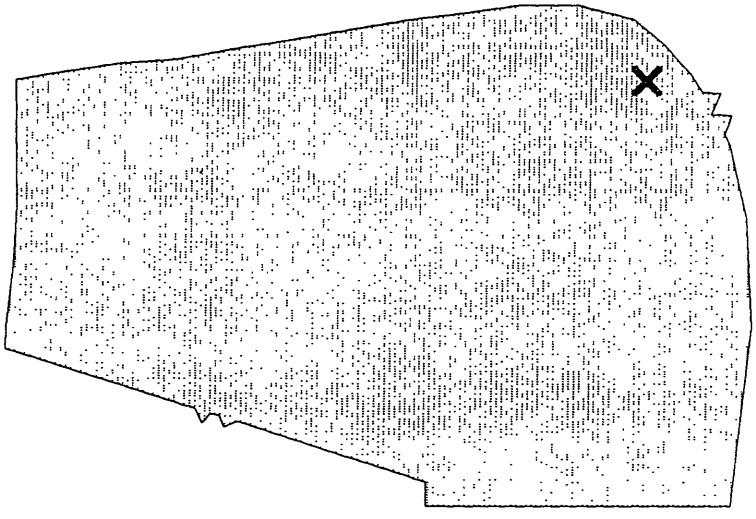


Figure 5-5a: Pants pocket piece with the first picker location marked. The picker is placed within one inch from the edge of the piece based on the picker to edge criterion.

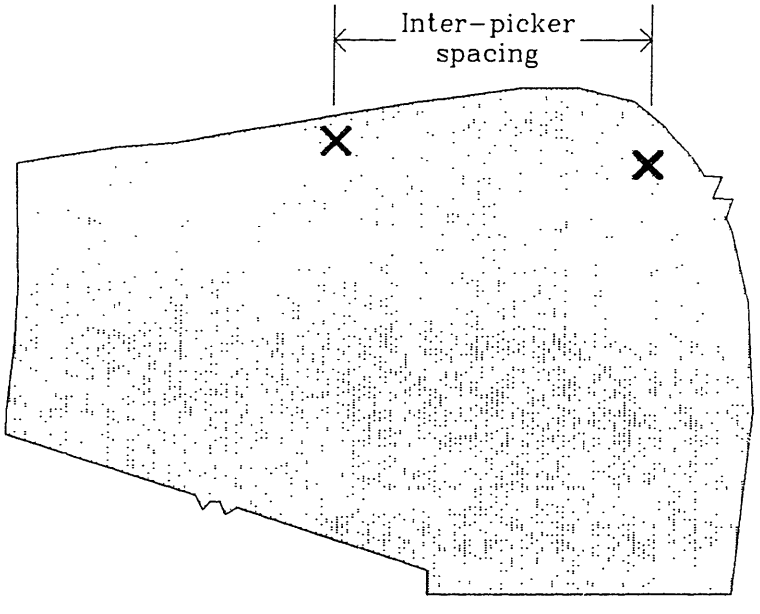


Figure 5-5b: Second picker location chosen based on inter-picker spacing criterion. The second picker is placed about six inches to the left of the first picker.

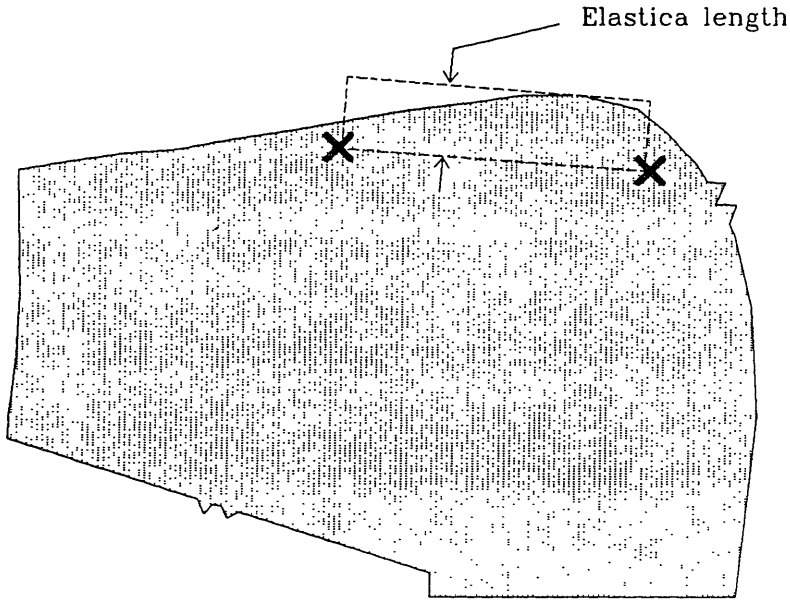


Figure 5-6a: A rectangular area is drawn to see if the picker to edge criterion is met. The criterion is not met because the width of the area is greater than the picker-to-edge criterion.

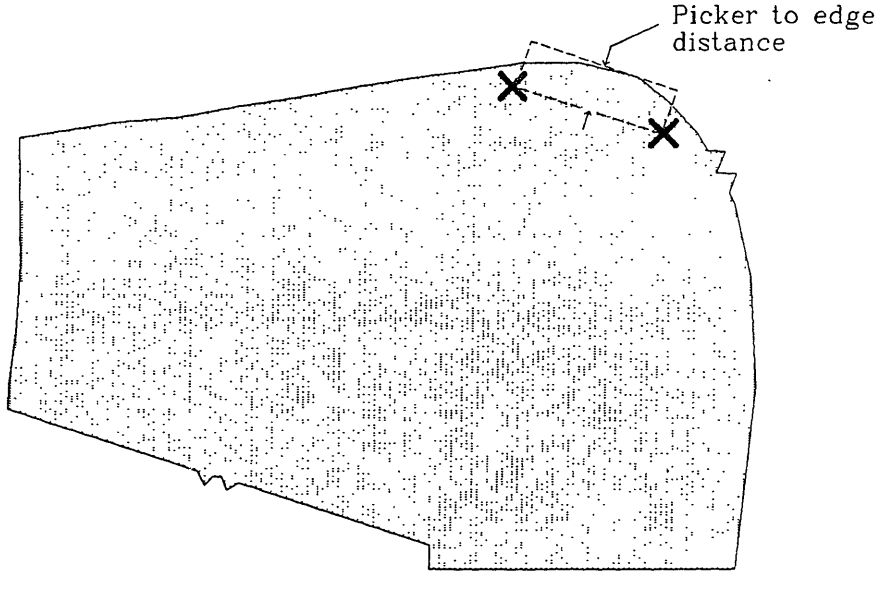
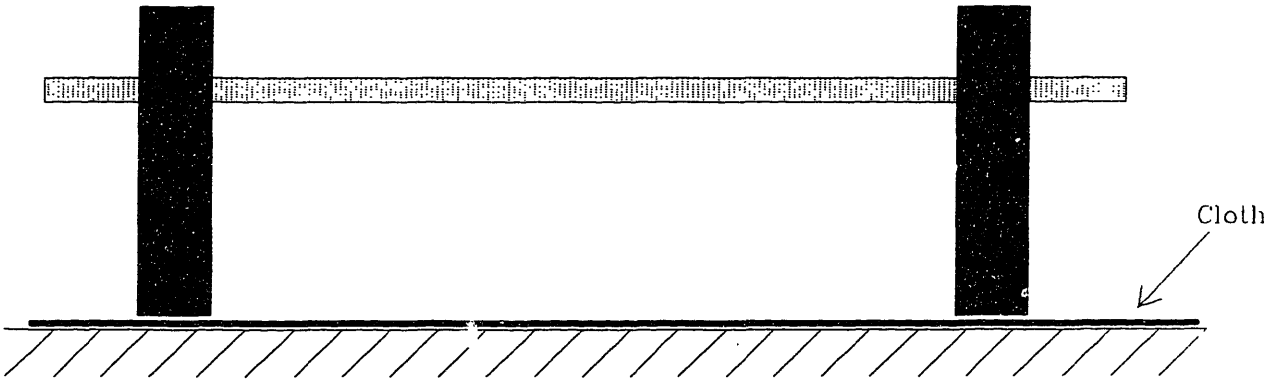


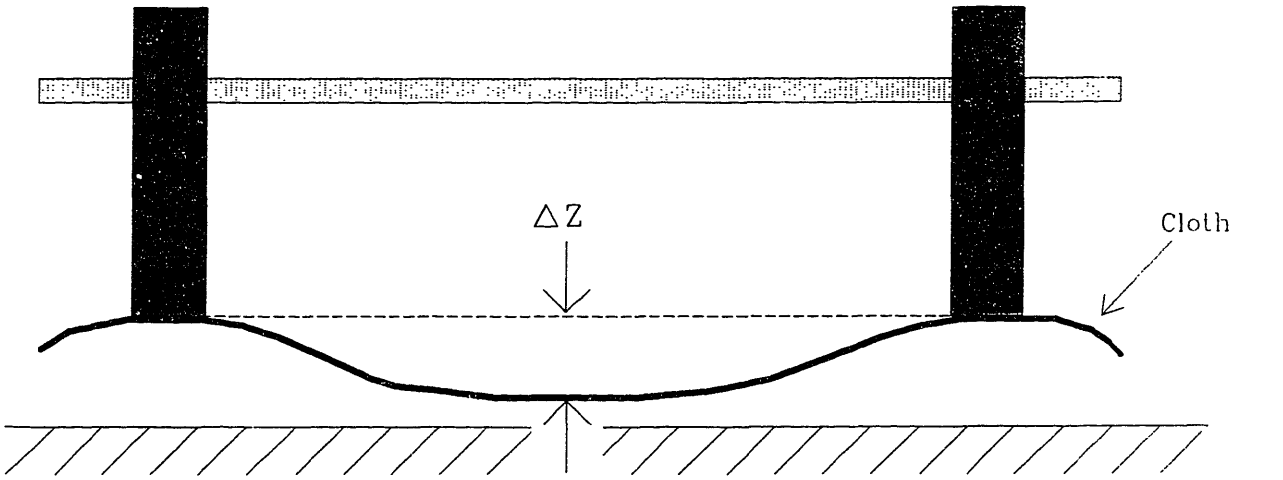
Figure 5-6b: The position of the second picker is adjusted so that the width of the rectangle falls within the picker-to-edge criterion.

left is moved up to the edge, the picker-to-edge criterion cannot be met. The problem is that the curvature of the piece requires that the pickers be placed closer together than the inter-picker spacing criterion, or else the picker-to-edge criterion cannot be met. A pair of picker locations satisfying the picker-to-edge criterion are shown in Figure 5-6b.

Despite the fact that the inter-picker spacing criterion is occasionally not needed, it is needed in general, thus one must still be developed. The inter-picker spacing criterion does not need to be very sophisticated. As previously stated, the goal is to limit the deflections of the edge of the cloth between pickers. Perhaps the simplest way to do this is to measure the alignment accuracy of a rectangular piece of cloth, picked up at two points. The alignment accuracy, A, is defined as the distance between the deflected edge of a piece of cloth and where it would be had it not been deflected. A test apparatus could consist of two adhesive pickers with adjustable spacing as depicted in Figure 5-7. The test apparatus would be used to pick up a rectangular piece of cloth. This rectangular piece of cloth would then be placed down on top of another rectangular piece of cloth. With the two piece of cloth



Test pick up device with adhesive on bottom of the two end pieces. The spacing between the two end pieces is adjustable.



The value of ΔZ is measured from the lowest point. The practical limit of this value will be determined empirically.

Figure 5-7: Test Apparatus for Determining Cloth Deflections Between Pickers

together, the edge alignment, A, is measured. The inter-picker spacing at which satisfactory edge alignment can be achieved is the maximum inter-picker spacing.

5.3 Discussion

Two criteria have been developed so far; the picker-to-edge criterion and the inter-picker spacing criterion. These criteria are necessary for the expert system to place pickers in appropriate locations. The picker-to-edge criterion prevents the edge of the cloth from being folded under. This criterion was developed based on an elastica model in Chapter 4. To apply this criterion to real pieces of cloth, rectangular regions between pickers are defined, and the width of these regions must meet the picker-to-edge criterion. The inter-picker spacing criterion prevents excessive shear deflections between pickers. These shear deflections between pickers cause poor edge alignment, and hence a poorly finished seam. The use of these rules is fundamental to the picker placement expert system to be discussed in the next chapter.

6 Expert System for Picker Locations

6.1 Introduction

The ultimate goal of this thesis is to develop the foundation of an expert system for determining picker locations. The previous chapters presented the various aspects involved in determining picker locations. Chapter 2 presented the basic hardware; Chapter 3 demonstrated how to automatically assembly pants; Chapter 4 derived a rule for how close to place pickers to the cloth edge; and Chapter 5 presented a method to establish the criterion for inter-picker spacing. Now, we are ready to consolidate this information and demonstrate how the expert system would determine appropriate picker locations. Some of the pants pieces from chapter 3 will be used as examples.

6.2 Picker Placement Expert System

The picker placement expert system can be considered as consisting of three different modules: user interface, inference engine, and knowledge base as shown in Figure 6-1. The knowledge base contains the rules for picker placement, e.g. picker-to-edge distance, and it also contains a catalog of cloth properties. The user interface is made up of the user input and the computer output. The user input to the

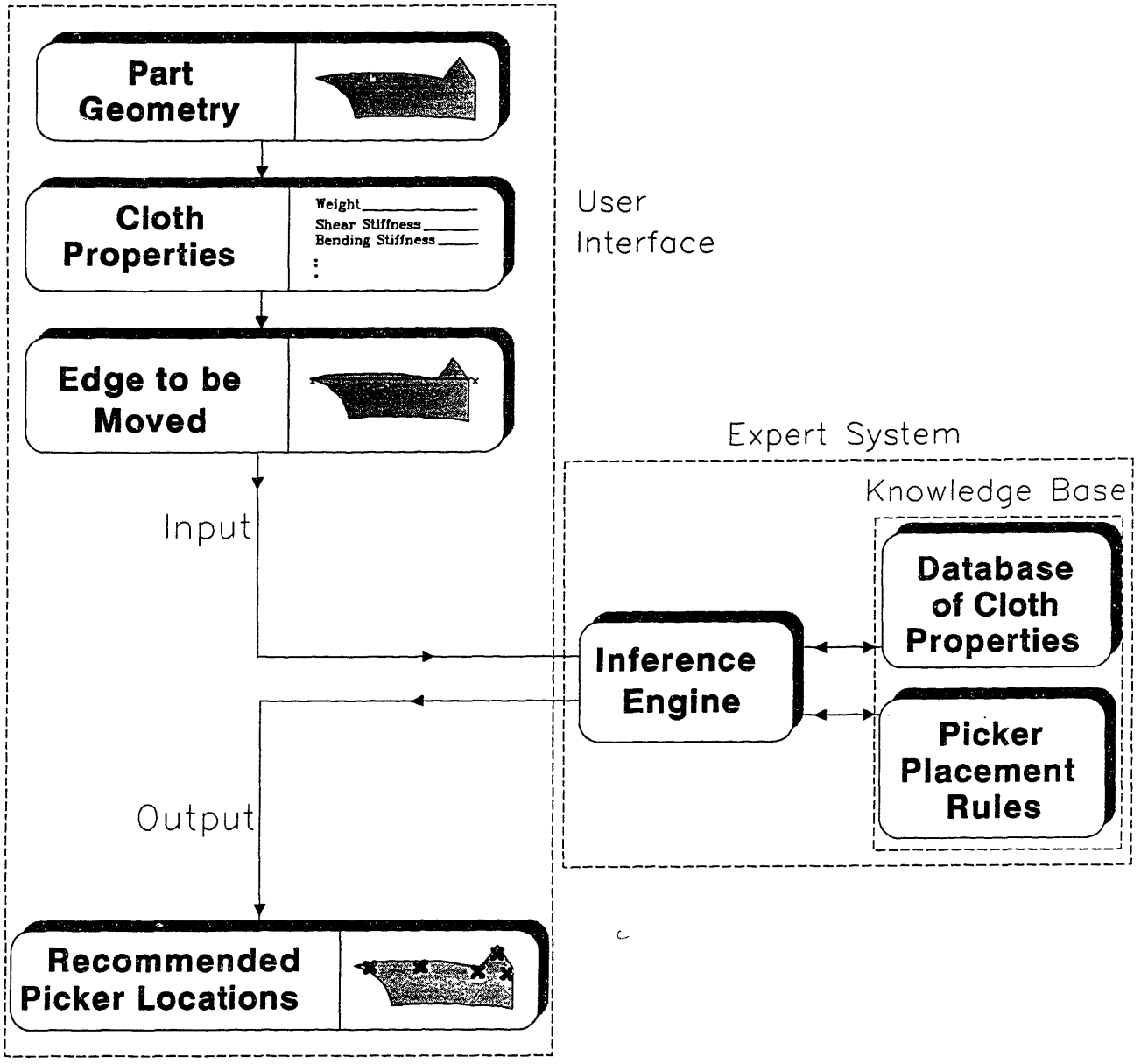


Figure 6-1: Schematic of Picker Placement Expert System.

expert system will be the part geometry, cloth properties, and the edge to be picked up. The output will be a suggested set of picker locations.

The knowledge base is the backbone of the picker placement expert system. It provides the basis for the inference engine to make decisions. This will be discussed in detail later. Part of the knowledge base is a catalogue of cloth properties. This catalogue relieves the user of the burden of repeatedly inputting cloth properties. Instead of entering in weight, bending stiffness, etc., the user can simply specify Milliken wool number 567, for example. The knowledge base will then supply the material properties for the chosen fabric.

The user input is mostly graphical in nature. For inputting part geometries, a vision system may be employed. In this arrangement, the work-piece is laid out in front of the vision system, which then digitizes the image and inputs it into the expert system. The expert system then shows the user a picture of the piece, and asks him to define the edge to be moved. An easy way to define the edge is with two points. Using a mouse or other pointing device, the user selects two points on the screen. Then the expert system highlights the edge selected, and ask the user if that is in

fact the edge to be moved. One other input that the user needs to supply is the fabric properties. The computer supplies a blank fabric properties input worksheet and the user has to fill it in. Once the data is entered, the user can give the data set a name so that it can be stored in the knowledge base.

Once the initial interaction between the user and the expert system is done, the system will determine and output the suggested locations of the pickers. The expert system puts X's at the locations where the pickers are to be placed. A more advanced form of output would be to have the expert system actually pick up the piece of cloth, via direct interface to the picker placement control system. In the latter case, the expert system would have to be capable of writing robot controller code to move the manipulator. These more advanced capabilities will not be considered here.

It should be noted that the amount of work involved in building an expert system, as just described, is quite large although there is no apparent fundamental reason that it cannot be done. (In fact such a system is being developed at Draper Labs as this thesis is written.) Before the expert system can be put together, there are a number of

interfaces that must be developed. The computer must have a graphical user interface capable of displaying images of cloth parts on the screen. A vision system with a "frame grabber" board and the appropriate software is needed to take "snapshots" of the cloth pieces. If a robot is going to be used as the output, then a robot interface would be needed.

Vision software "tools" are needed for the expert system. Algorithms for edge finding, break point finding, scaling, rotation, etc. would be needed. For the expert system to carry out a simple instruction, like place a picker one inch from the edge, it must find the edge of a bit mapped image and rotate the image so that a tangent to the edge is parallel to the axis of the vision system. Given the current state of the art in computers and vision systems, none of these steps is new. However, a great deal of development effort must be spent to realize such a system, which is beyond the scope of this thesis.

6.3 Rules for Picker Placement Expert System

The expert system works from a list of rules. Two of the rules for picker placement, picker-to-edge rule and inter-picker spacing rule, have been described in the last two chapters. In addition, one more simple rule must now be added. This new rule simply reflects the fact that most pickers do not work very close to the edge of the cloth. A minimum distance between the center line of the picker and the edge of the cloth must be specified. For the Walton pickers this distance is about 0.5 inch (1.27 cm). Other pickers, such as an adhesive picker may have no minimum distance.

A summary of the picker placement rules, written in plain English, is as follows.

Rule 1: Using the coefficient of friction of the folding surface, cloth stiffness, cloth weight, the elastica mode graph (Figure 4-10), and $L = \beta^{1/3} \cdot (E \cdot I/w)^{1/3}$ (Equation 4-8), determine the length L which is the maximum allowable distance between a picker and the cloth edge. The perpendicular distance that starts on a line drawn between two pickers and ends on the cloth edge, must not exceed the length L. When a picker does not have pickers on both sides of it (i.e. an end picker) then the picker must also be within the distance L from the cloth edge at the corner of the edge to be moved.

Rule 2: The maximum inter-picker spacing L_i is the inter-picker spacing at which a rectangular piece of cloth, picked up by the test fixture (Figure 5-7), is sufficiently stiff to achieve the alignment accuracy A. The distance between two adjacent pickers must not exceed L_i.

Rule 3: The distance between the center line of a picker and the cloth edge must not be less than the minimum functional distance L_{min}. The value of L_{min} is determined experimentally. The distance between a picker and the nearest edge must, under no circumstance, be less than L_{min}.

These rules will be clarified by examples in the following section. If these rules were actually written for an expert system, they would be more complicated and less like English. The expert system cannot *interpret* the rules, it can only follow them. To totally clarify these three rules to the expert system would probably require more rules and even sub-rules.

6.4 Picker Placement Examples

For the first example, consider trying to perform the pickup operation of the folding station 9 from Chapter 3. The pants back (V) must be picked up and placed on top of the pants front (P) so that the outseam is aligned, see Figure 6-2. The outseam edge of pants back (V) must be picked up off the table before the edge can be aligned. To input the cloth geometry, the user places the piece to be moved on top of the folding table. The vision system takes a picture of the piece and displays it on the users computer screen. Then the computer asks the user to select the edge to be moved, see Figure 6-3. The user selects the edge to be moved by marking the two corner points of the outseam on the computer screen, see Figure 6-3. The computer responds by highlighting the outseam and asking the user to confirm that this is the edge to be moved, see Figure 6-4.

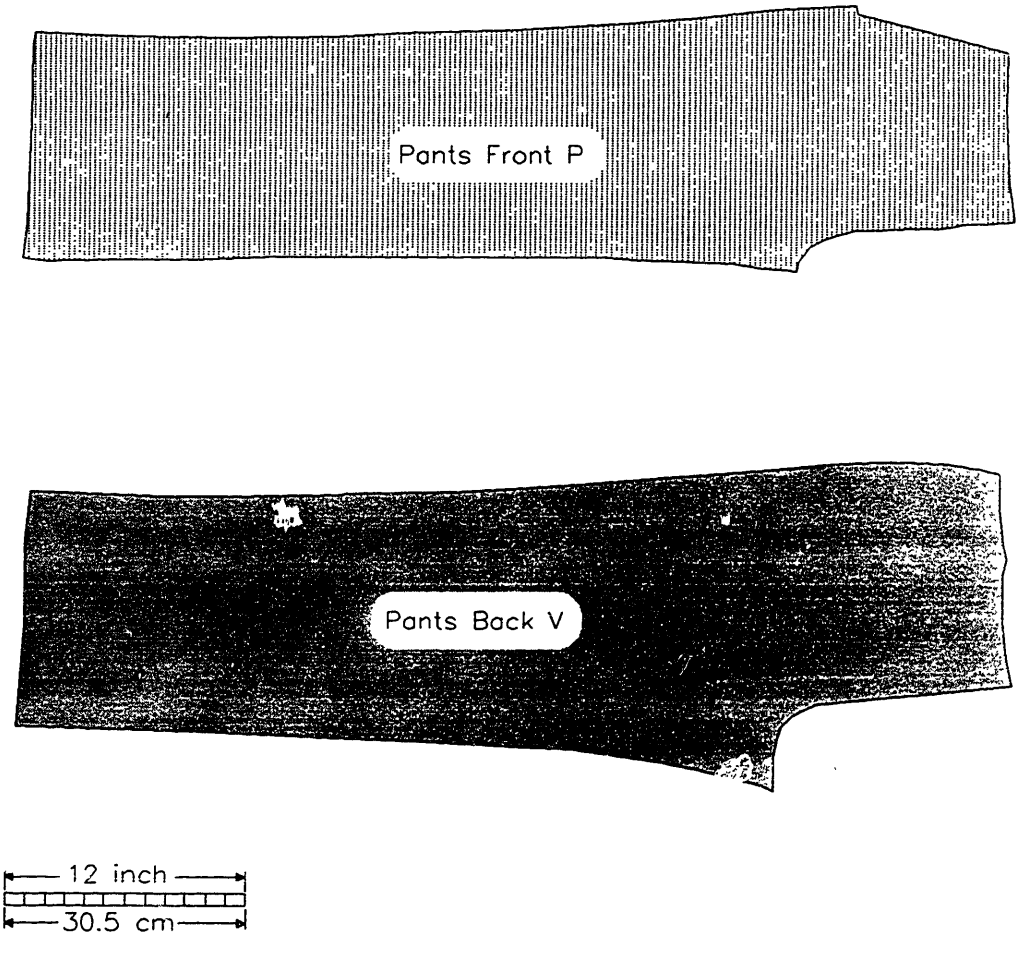


Figure 6-2: Two pant pieces prior to alignment. The two edges (side seams) must be aligned before sewing.

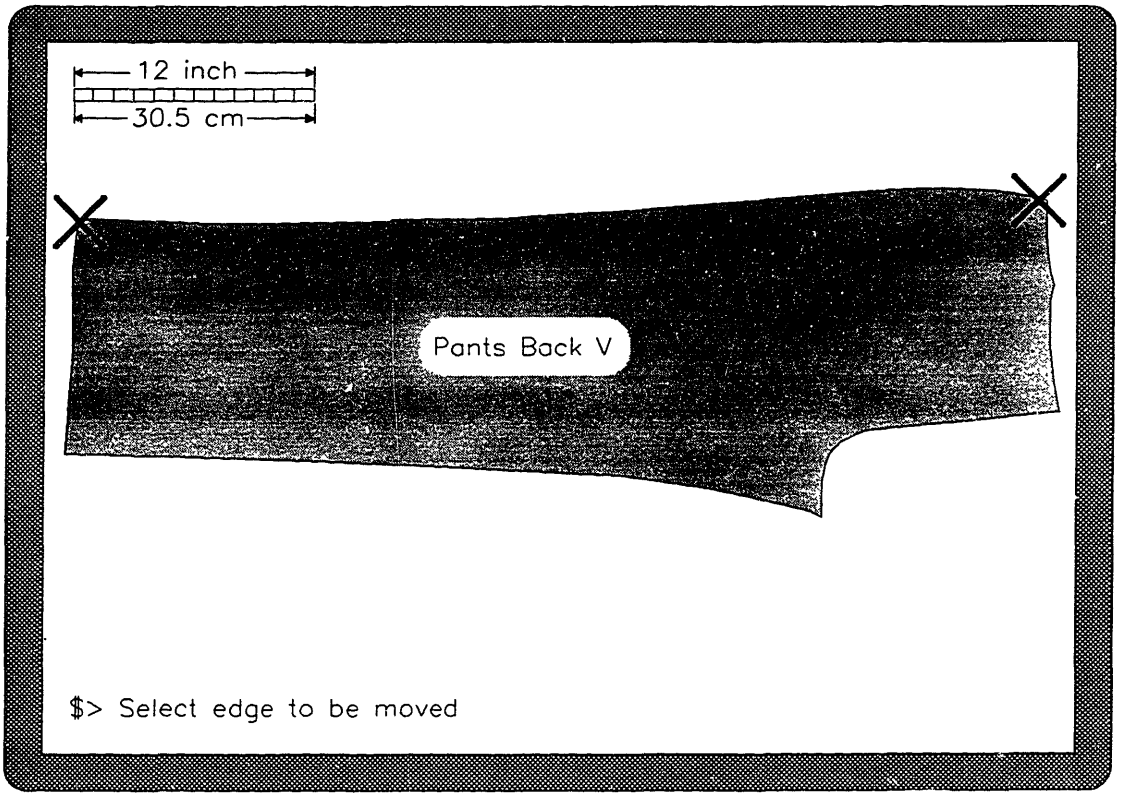


Figure 6-3: User selects the edge to be moved by marking two points on a computer image of the part.

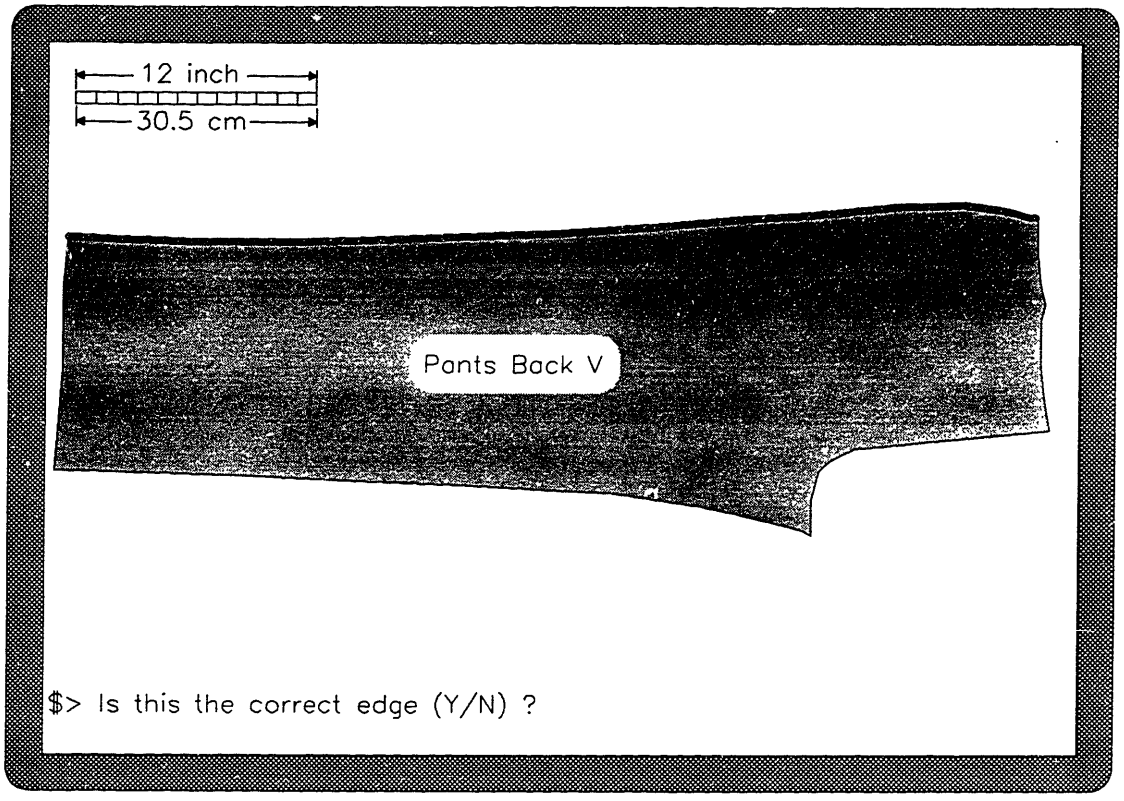


Figure 6-4: The computer responds by highlighting the selected edge.

In order to apply the rules the expert system asks the user to input the properties of the cloth. For Rule 1, the expert system needs to know the weight and stiffness of the cloth as well as the coefficient of friction of the folding surface. For the suit pants material the weight is 0.000201 lbf/in (0.000036 kg/cm) for a one inch width, the stiffness is 0.000086 $lbf \cdot in^2$ (0.00025 $kg \cdot in^2$) for a one inch width. The coefficient of friction of cloth on the folding surface is about 1.0. Using (Figure 4-10), $\mu = 1.0$, $\beta^{1/3} = 1.8$ and $L = \beta^{1/3} \cdot (E \cdot I/w)^{1/3}$, the value of L is found to be 1.36 inches (3.45 cm), see Figure 6-5. For Rule 2, the expert system asks the user how far apart the pickers can be and still achieve satisfactory edge alignment. Previous tests on the suit pant material yielded the following information.

Inter-picker Spacing	Drape ΔZ	Alignment Accuracy
29.0 inches	1.0 inch	0.5 inch
15.0 inches	0.3 inch	0.1 inch
8.0 inches	0.1 inch	≈0.0 inch

For an inter-picker spacing of 8 inches (20 cm) or less there is no significant alignment error. The user enters 8 inches (20 cm) as the maximum inter-picker spacing. Finally for Rule 3, the expert system asks what type of picker is being used. The user enters Walton as the picker to be

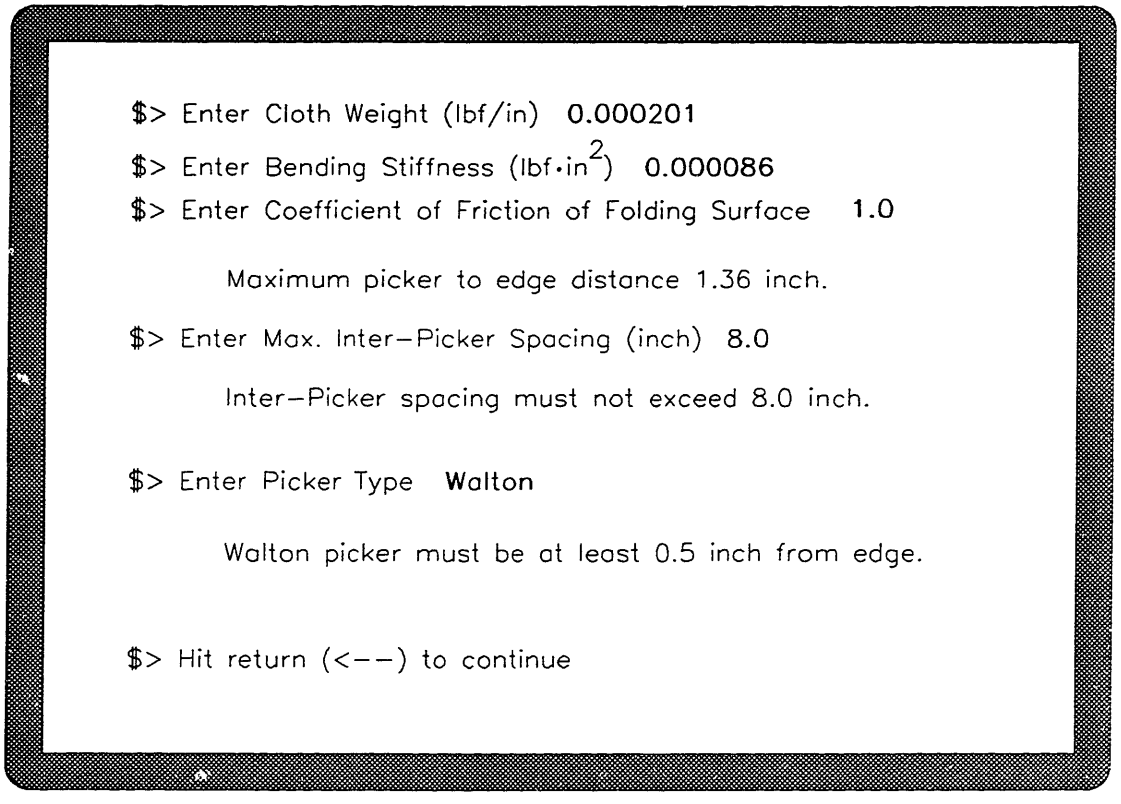


Figure 6-5: Expert system prompts user to input cloth data.

used. The expert system responds that Walton pickers must be at least 0.5 inch (1.27 cm) from the edge of the cloth. This information came from an internal knowledge base of picker information. At this point the user has entered sufficient information for the expert system to proceed with finding picker locations.

The expert system starts placing pickers at the upper left hand corner of the pants back (V), see Figure 6-6. For the first picker, it must only satisfy Rules 1 and 3. Since there are no other restrictions, the expert system chooses to place picker #1 at a distance of 1.36 inches (3.54 cm) from both edges. The next picker placement must satisfy all three rules. Due to the lack of curvature of the edge, picker #2 can be placed the full 8 inches (20 cm) away from picker #1. It can also be placed 1.36 inches (3.54 cm) in from the edge. Pickers #3, #4, and #5 can all be placed in the same manner as picker #2. Picker #6 must be placed closer to the edge than the other pickers in order to satisfy Rule 1, see Figure 6-7. Picker #6 is placed 1.1 inch (2.8 cm) in from the edge. Note that 1.1 inch is less than 1.36 inch, as required by Rule 1, but greater than 0.5 inch, as required by Rule 3. Picker #7 is placed 1.2 inch (3.0 cm) in from the edge for the same reason as picker #6. Having finished the picker placement, the expert system

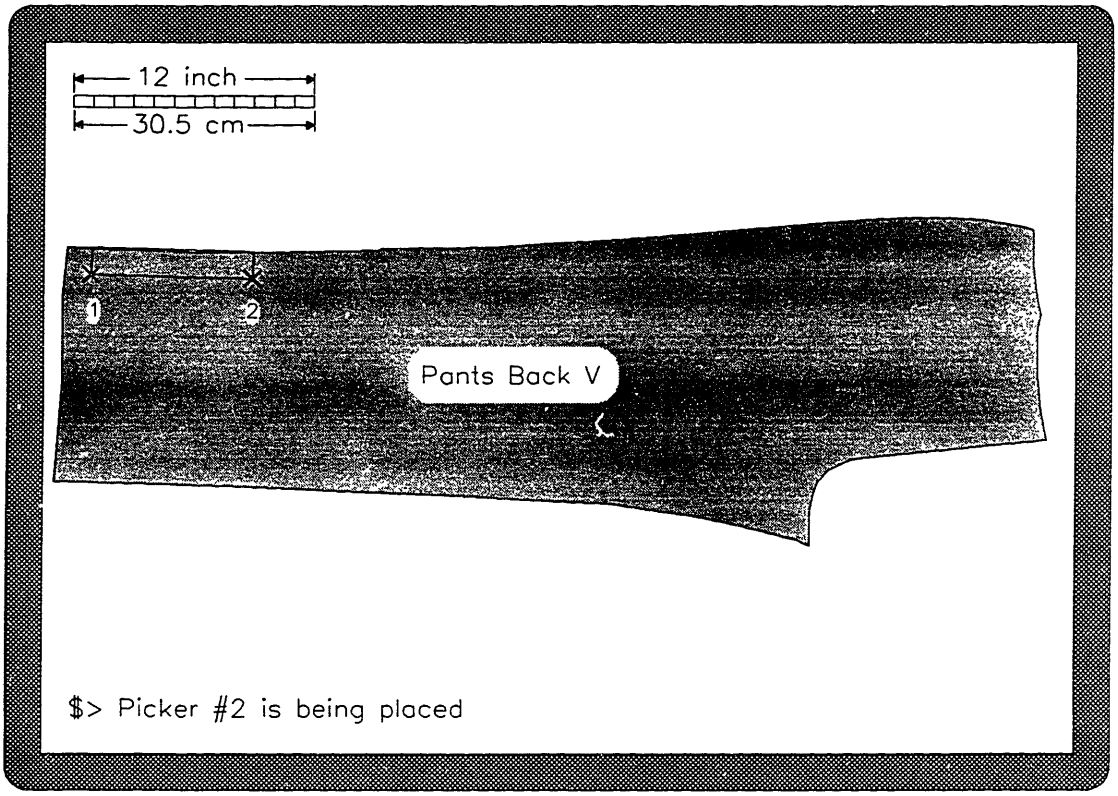


Figure 6-6: The first two pickers are placed 1.36 inch from the edge of the cloth. The pickers are a full 8 inches apart due to the large curvature of the piece.

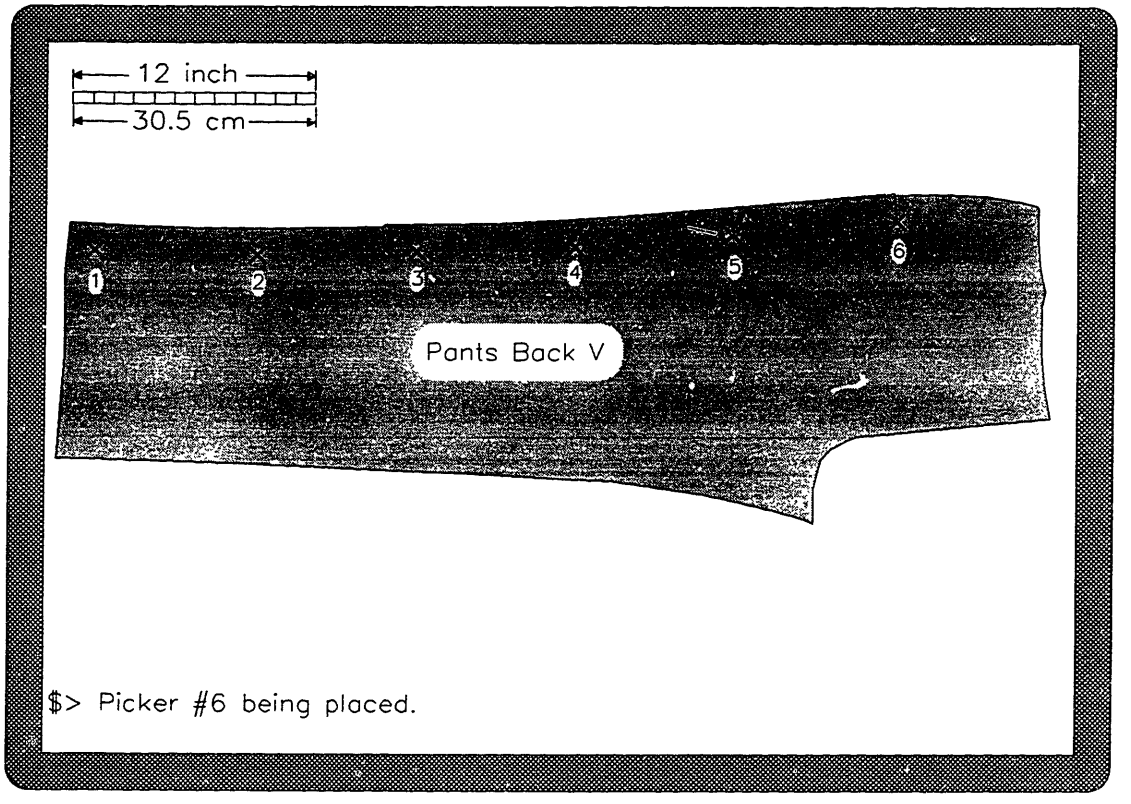


Figure 6-7: The computer places picker #6 at 1.1 inch from the edge due to the decreased radius of the edge.

shows the user the locations of all the pickers, see Figure 6-8.

For the second example of picker placement, consider aligning the inseam of pants back (V) and pants front (P). The inseam of pants back (V) must be picked up, moved over, and aligned with the inseam of pants front (P). Since the information about the cloth properties has already been entered, the expert system does not prompt the user to enter this information. As before, the user places the piece to be picked-up on the folding table. The vision system takes a picture and displays it on the computer screen. Then, the expert system prompts the user to select the edge to be moved. The user marks the inseam with two X's, the computer responds by highlighting the selected edge and asking the user if that is the edge to be moved.

Now the expert system goes right to work. Pickers will be placed starting from the lower left hand corner, see Figure 6-9. Picker #1 is placed 1.36 inches (3.54 cm) in from both pants bottom and the inseam. In accordance with Rules 1 and 3, pickers #2 through #5 are placed 8 inches (20 cm) apart and 1.36 inches (3.54 cm) in from the edge. The placement of picker #6 is not obvious. If picker #6 is

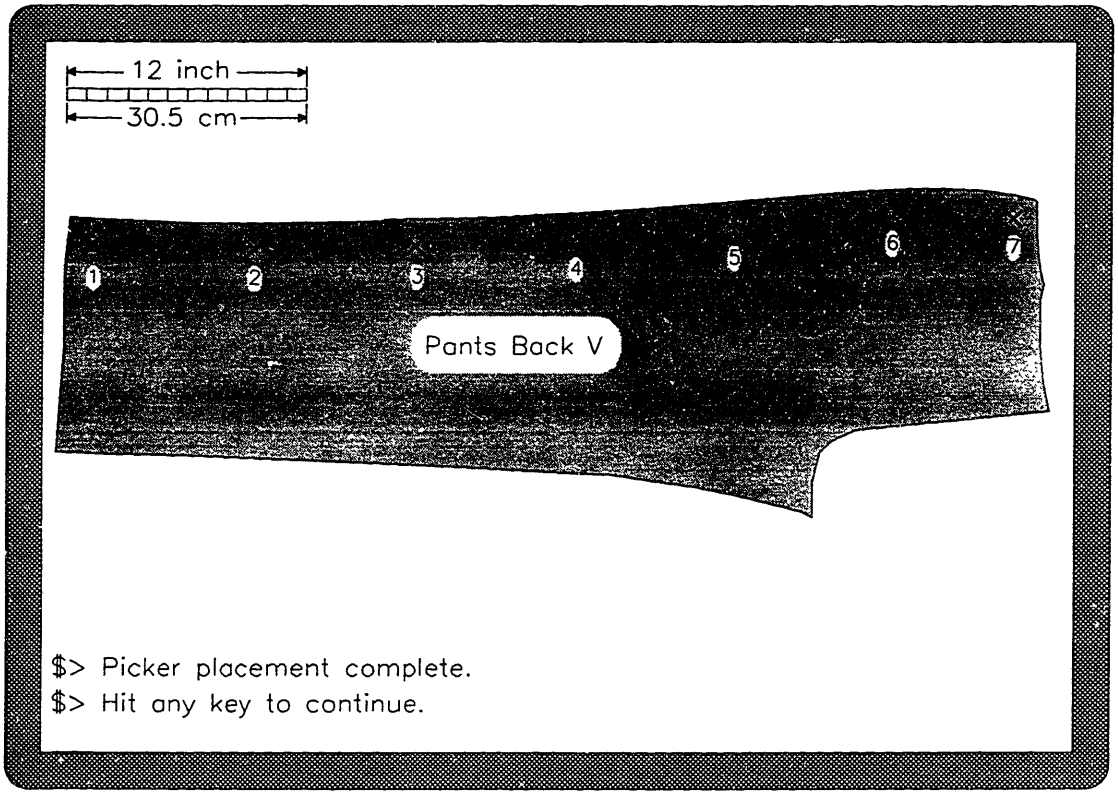


Figure 6-8: All the pickers locations for the edge to be moved have been choosen by the expert system.

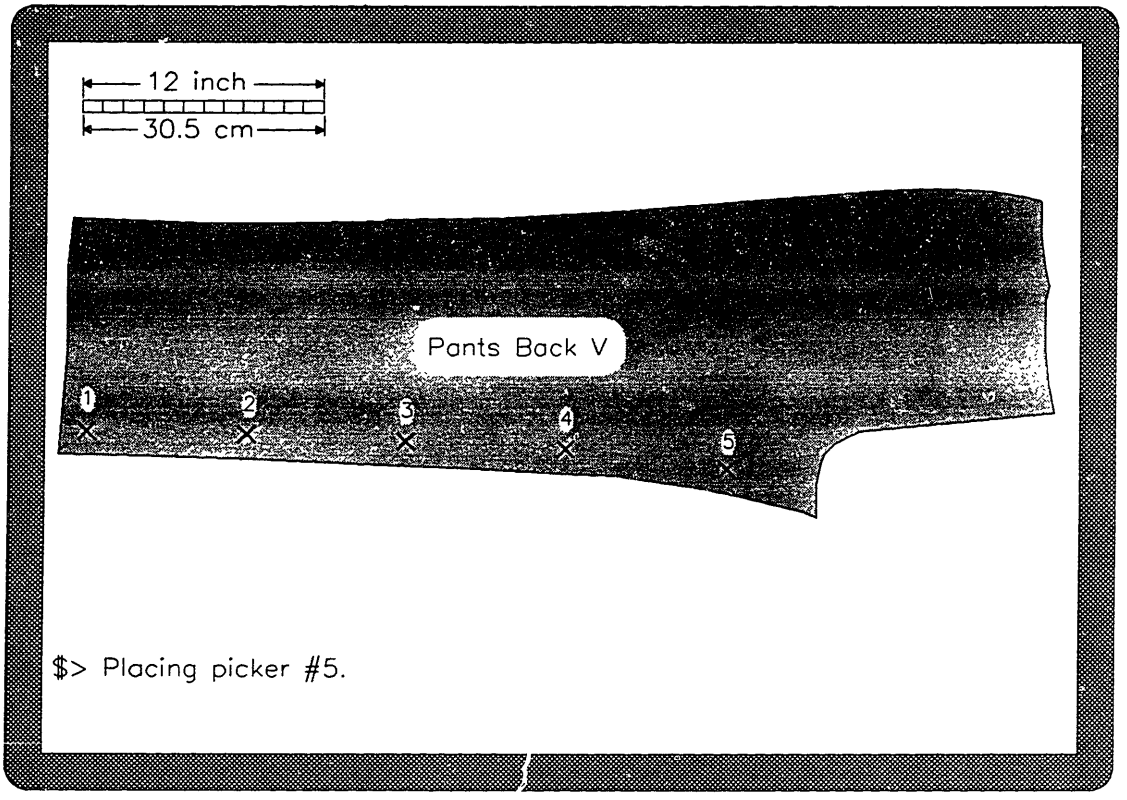


Figure 6-9: First 5 pickers are placed on the inseam of pants back (V).

placed 8 inches (20 cm) away, from picker #5, then Rule 1 is violated, see Figure 6-10. To satisfy Rule 1, picker #6 must be placed 1.36 inches (3.54 cm) from the point that will be the crotch of the pants, see Figure 6-11. Although this means placing pickers #5 and #6 only 2.5 inches (6.35 cm) apart, it is necessary to satisfy all the rules. Picker #7 is placed 8 inches (20 cm) from picker #6 and 1.36 inches (3.54 cm) in from the edge. Picker #8 is placed 1.36 inches (3.54 cm) in from both the inseam and the top of the pants. This concludes the placement of the pickers for the inseam of pants back (V), see Figure 6-12.

For the third and final example, consider trying to pick up the pant's front pocket piece (R). This piece is considerably smaller than the pant's front and back. Several folding operations, from chapter 3, involve picking up the inside edge of the front pocket (S). To find picker locations for the inside edge of the front pocket, the user places the pocket on the folding table, the vision system takes a picture, and then the expert system displaces and image of the pocket piece on the screen. The user selects the edge to be moved, the computer highlights the edge and asks if it is the edge to be moved. The expert system starts placing pickers at the upper left hand corner of the piece, see Figure 6-13. The picker #1 is placed 1.36 inch

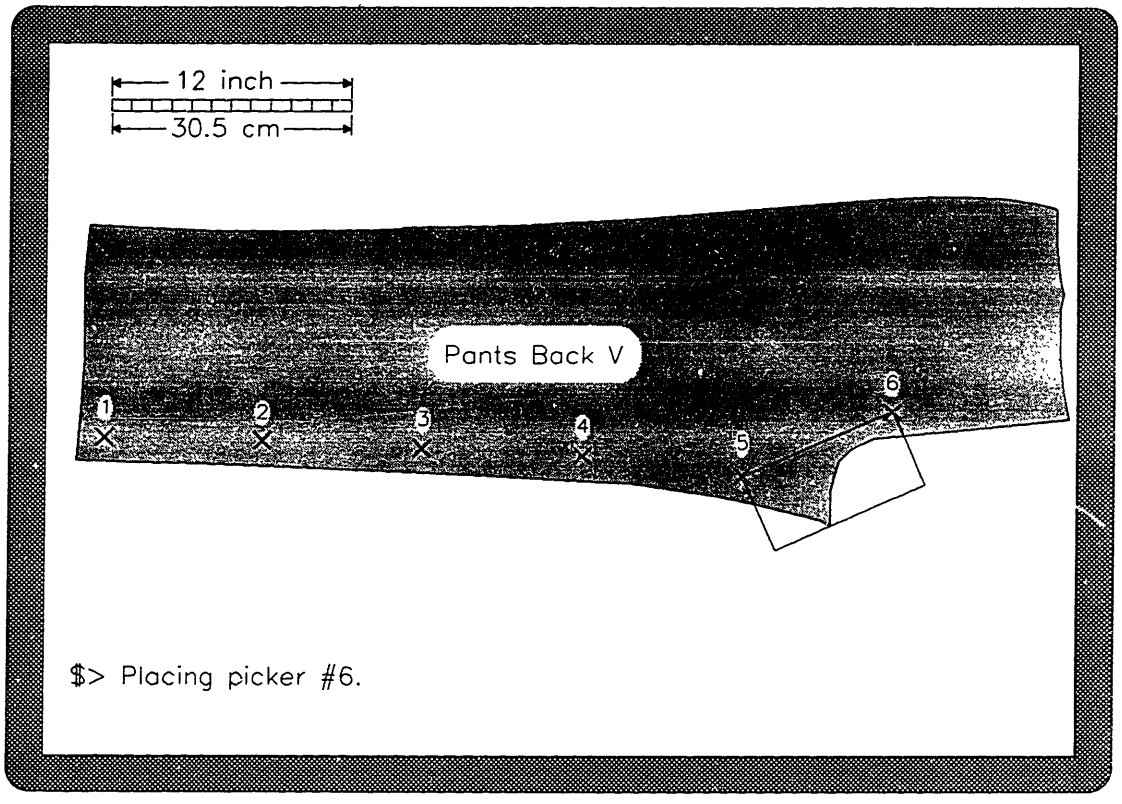


Figure 6-10: An attempt is made to place picker #6 8 inches from picker #5. A rectangle is drawn to show that this placement of picker #6 violates rule 1.

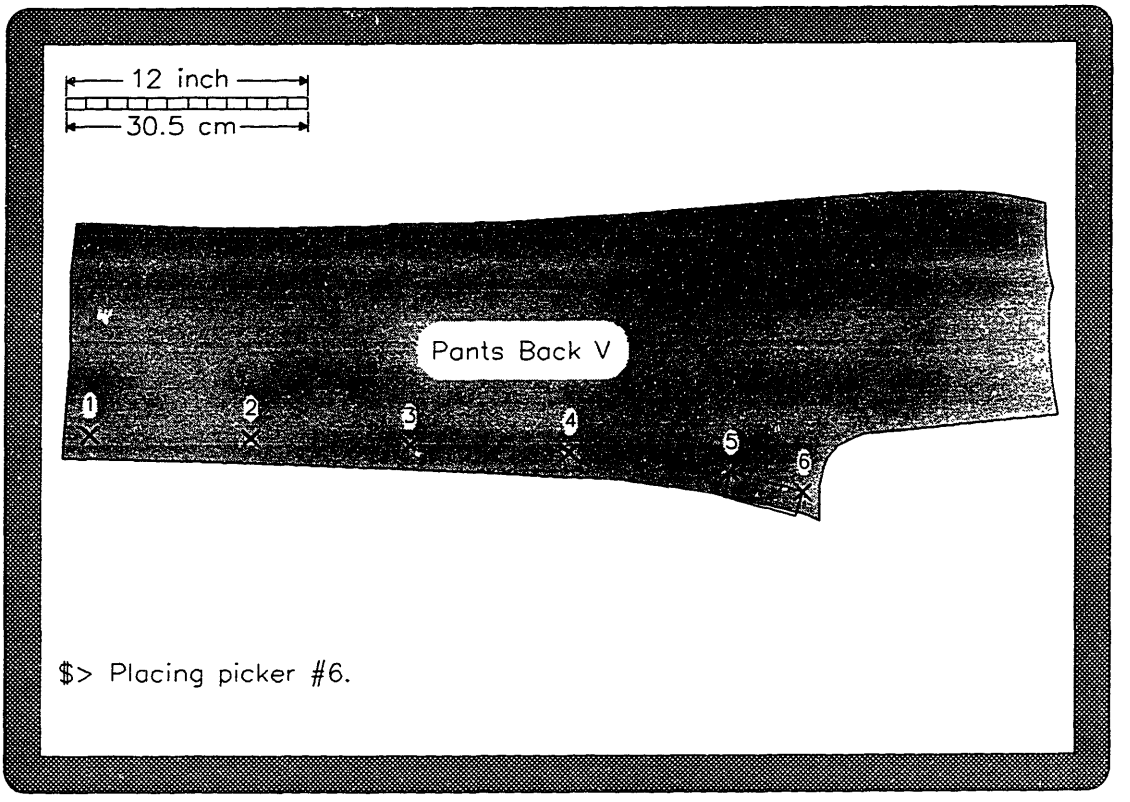


Figure 6-11: Picker #6 is placed within 1.36 inch from the corner. This is only 2.5 inches from picker #5 but it is necessary to satisfy rule 1.

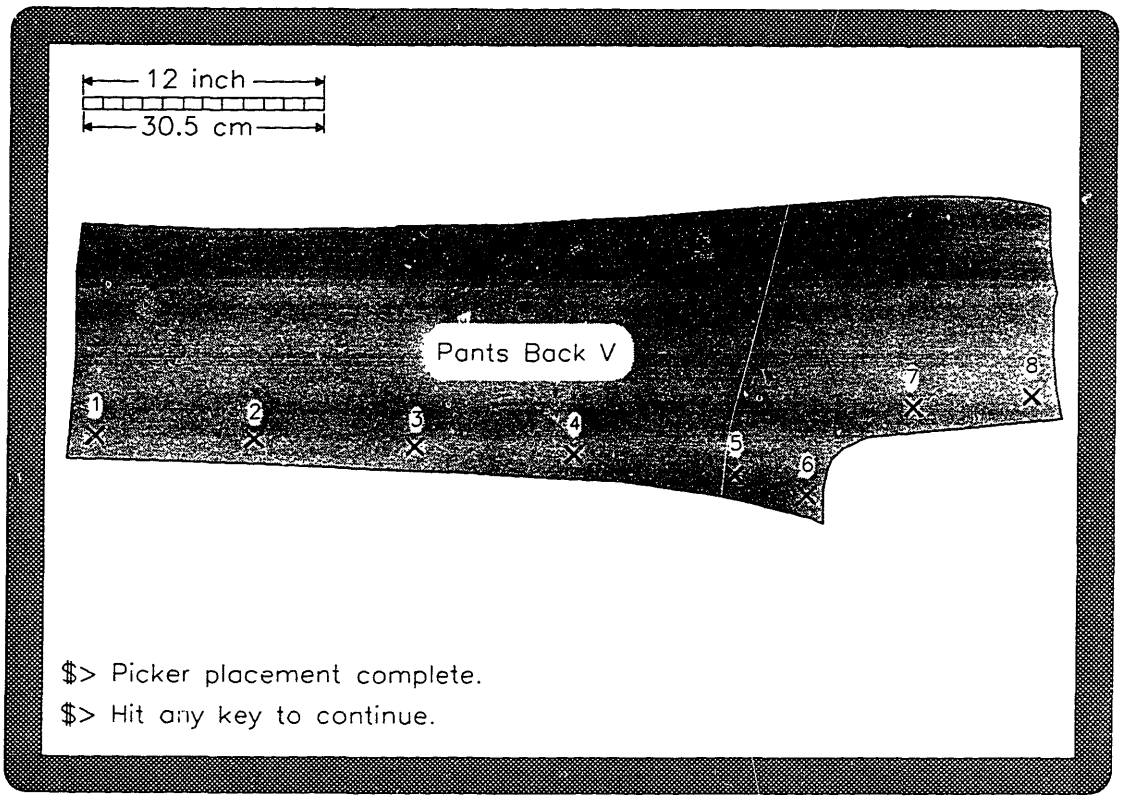


Figure 6-12: All the picker location for the inseam are choosen.

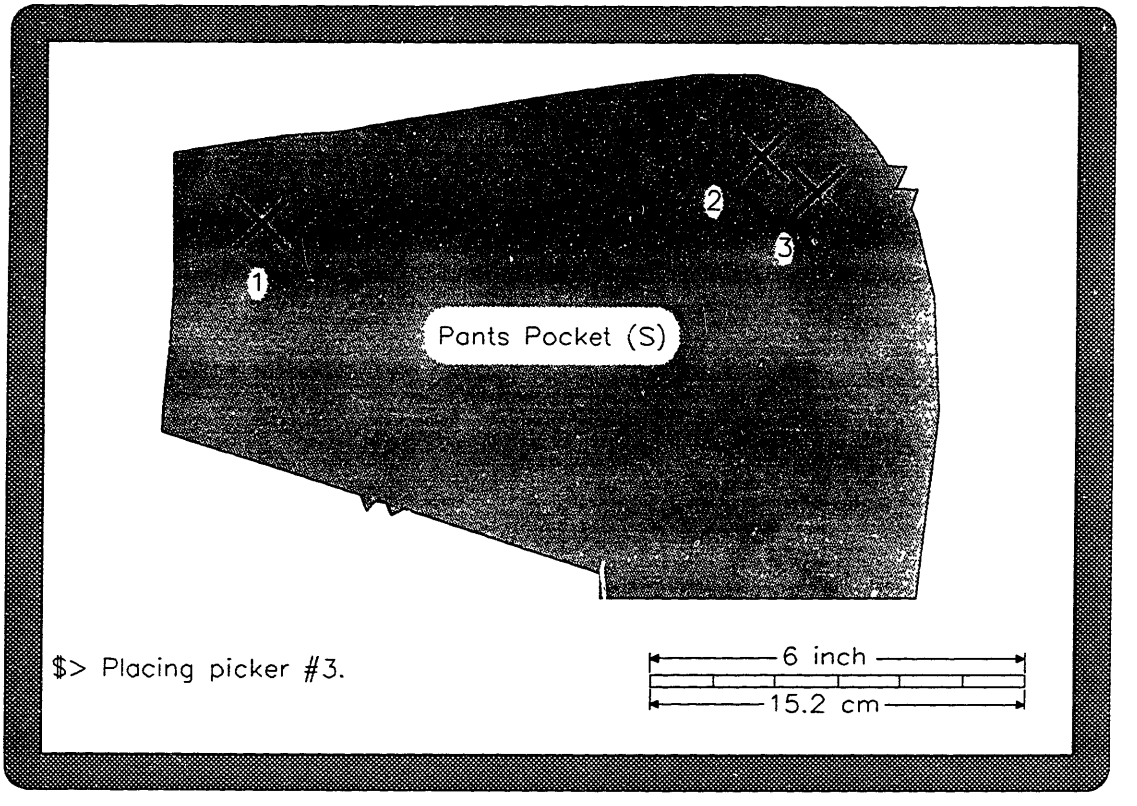


Figure 6-13: Picker #2 is placed 1.36 inch from the edge and 8 inches from picker #1. Due to the high radius of curvature, picker #3 must be placed placed close to picker #2 to satisfy rule 1.

(3.54 cm) from both edges. Picker #2 is placed 8 inches (20 cm) from picker #1 and 1.36 inch (3.54 cm) from the edge. Now there is a slight problem. Due to the small radius of curvature, picker #3 must be placed very close to picker #1 in order not to violate Rule 1, see Figure 6-13. This problem can be avoided by placing picker #2 0.5 inch (1.27 cm) from the edge, see Figure 6-14. By doing this picker #3 can be placed much further away from picker #2, thus reducing the total number of pickers needed. Picker #4 is placed 1.36 inch (3.54 cm) from both edges, see Figure 6-15.

6.5 Adding New Rules to Expert System

When developing an expert system it is necessary to turn thought processes that are "common sense" into rules. Consider the third example of picker placement given above. To use fewer pickers, picker #2 was placed close to the edge. For the author, this was simply "common sense." For the expert system to do the same operation requires a rule, a new rule. The rule might be expressed as follows.

Rule 4: When the radius of curvature is small place the pickers as close to the edge as allowed by Rule 3.

With this new rule, the expert system can make better choices of picker locations.

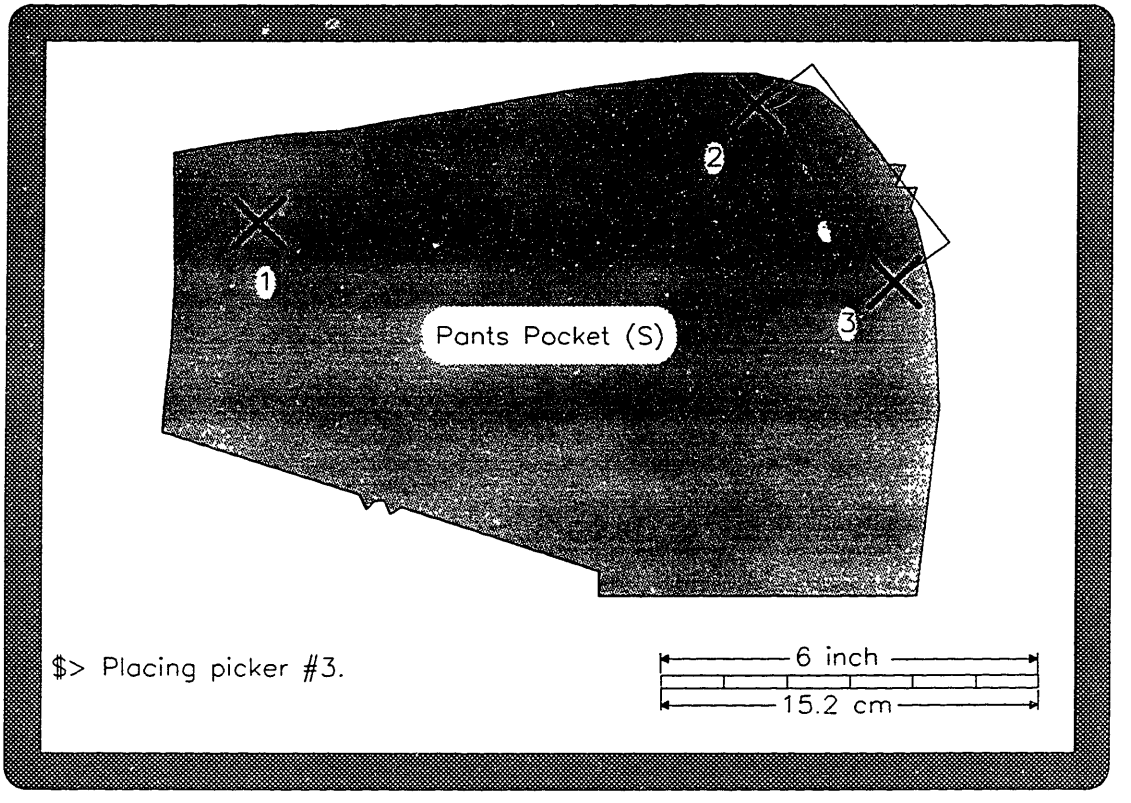


Figure 6-14: Picker #2 is placed 0.5 inch from the edge and 8 inches from picker #1. Picker #3 is placed 0.5 inch from the edge and as far away from picker #2 as rule 1 will allow.

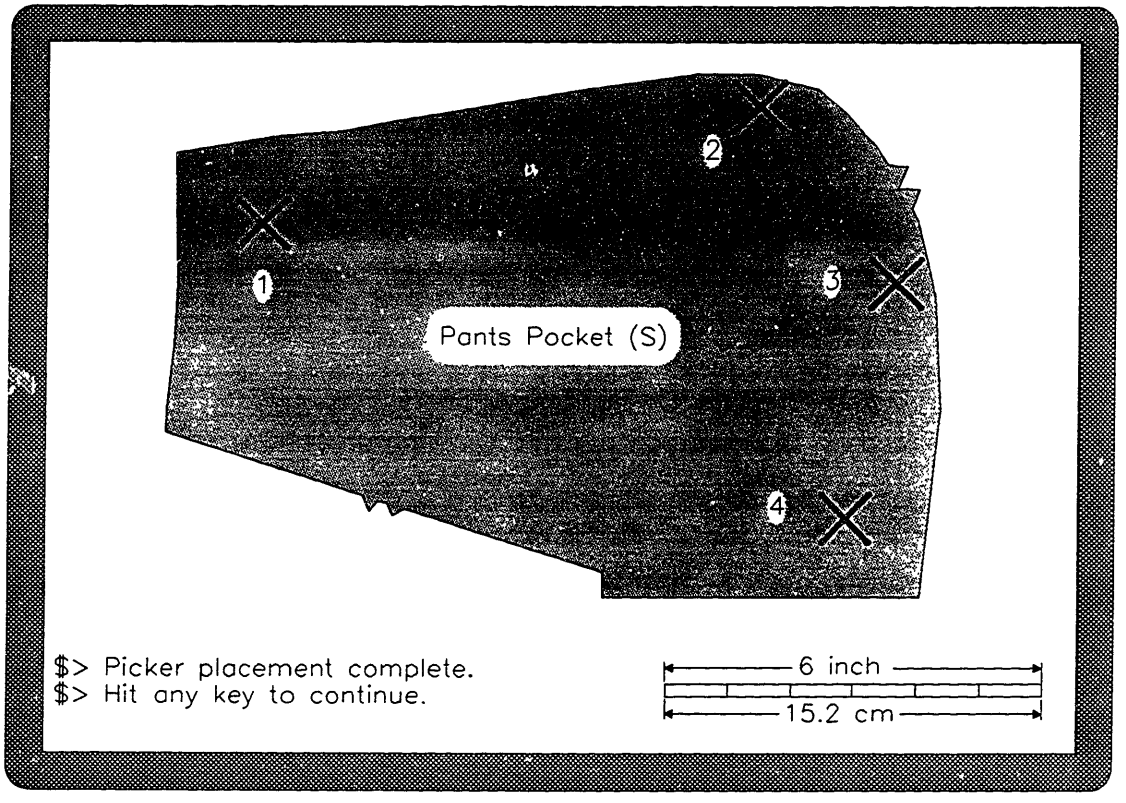


Figure 6-15: Picker placement for pants pocket piece (S) is complete.

Creating rules for an expert system is a dynamic process. Initially, rules are created based on obvious facts. Once the system is running, new rules are added based on how well the system performs. Again "common sense" must be taught to the expert system through the creation of new rules. Hence, the expert system developed must have the provision to acquire new knowledge and update its knowledge base.

6.6 Optimization of Picker Locations

The picker locations given in section 6.4 are by no means optimized. They are an acceptable set of picker location that could be improved. Consider Pickers #5 and #6 in Figure 6-11. These pickers should be spread out more and the spacing between pickers #1 through #5 should be reduced. This would yield better cloth control. No optimization was sought because picker locations are always limited by the flexibility of the end-effector and thus hardware limitations would overrule the optimization.

6.7 Hardware Limitations

As discussed in Chapter 2 there are different types of end-effectors with different limitation. For some end-effectors, the limitations can be quite small. For other end-effectors, the limitations can be a major consideration. This hardware limitation is fundamental to picker placement. This issue has been disregarded until now for two reasons: 1) there exists very few end-effectors thus little is know about the possible limitations of future end-effectors, and 2) part of the usefulness of the expert system is to help design end-effectors. The second point is very important. Consider trying to build all the end-effectors for the automated transfer line of chapter 3. The first step would be to give the expert system all the pieces that must be moved and ask it to find the picker locations. Based on the results of the picker locations an end-effector could be designed, e.g. if more pickers are generally needed at the ends of the end-effector and less in the middle, then the end-effector would be designed with more pickers on the ends. The expert system could be taught to confine picker locations to the limitations of the newly designed end-effector.

An important low cost end-effector is one where the pickers cannot be moved at all. This type of end-effector is used for a specific part geometry, e.g. the inseam of a

size 42 shirt sleeve. For this end-effector, the expert system could determine the picker locations and the end-effector would be built with the pickers just where the expert system wants them to be. This type of end-effector is only useful for high volume production where flexibility is not needed.

It is worth emphasizing the fact that fixed inter-picker spacing is not desirable. The current $(TC)^2$ end-effector uses fixed inter-picker spacing, and suffers because of it. With fixed inter-picker spacing the pickers must be placed very close together in order to have a picker near the location where it is needed. Fixed inter-picker spacing typically leads to too many pickers where they are not needed, and not enough where they are needed.

6.8 Conclusions

An expert system, given a number of rules, based on the physical behavior of cloth, is capable of determining a suitable set of picker locations. The two basic rules of 1) picker-to-edge rule and 2) inter-picker spacing rule; are primarily responsible for selecting a suitable set of picker locations. Additional rules are needed to specify other physical limitations, e.g. how close to the cloth edge a picker can be used. Three examples have been given to

demonstrate how the rules are used to determine picker locations. Additional examples of increasingly complex shapes would result in the need for new rules. These new rules would be added to the picker placement expert system knowledge base. The building of a knowledge base is a never-ending process.

7 Discussion of Picker Placement Method

This thesis is responsible for laying the foundations of a major improvement in flexible materials handling. Individuals that develop automated textile equipment rarely give much thought to the mechanical behavior of cloth. This was the case for the $(TC)^2$/Draper labs sleeve machine. Some simple testing of picker locations led to the development of the end-effector. Almost no consideration was given as to whether the system had too many pickers or too few. The work in this thesis demonstrates that it is possible to approach flexible material handling in a scientific manner. Implementation of the picker placement expert system, would drastically reduce the amount of trial-and-error that is frequently used to determine picker locations. The ultimate goal in flexible materials handling is to have values of machine parameters, e.g. picker positions, be determined by material properties, e.g. bending stiffness, and part geometry. By proceeding in this fashion, the engineering of flexible materials handling can be as sound as other engineering disciplines.

The development of a expert system for making decisions on flexible materials handling represents a new level of thinking in cloth handling problems. The idea is that for

areas of cloth handling where there is little "deep knowledge" (scientific), use "surface knowledge" (rules of thumb) to determine the answers. Where there is "deep knowledge", however, use science to determine the answer. An expert system is a computer program that can combine both deep and surface knowledge to culminate a powerful analytical tool. The use of this tool by engineers and designers, who are developing flexible materials handling equipment, can dramatically improve the development process and the final hardware.

The picker-to-edge criterion demonstrates deep knowledge of cloth behavior. The key question facing designers was how far away from the edge can picker be before the edge of the cloth is inadequately supported. The development and solution of an elastica model gave the answer to this fundamental problem. Experiment verification of the model gave the author the confidence to use this as part of the rules of picker placement. The two dimensional model is used as an approximation of the three dimensional problem of a piece of cloth being picked up by an end-effector. Therefore the knowledge about the two dimensional problem yield approximations for the three dimensional problem. These approximation are, however, miles ahead of

guessing. In fact, for piece with large radii of curvature the three dimensional problem approaches the two dimensional problem.

It is important to realize the true value of the picker placement expert system presented in this thesis. Its value goes beyond the ability to place pickers in appropriate locations. The real value is that it represents a new trend in working with flexible materials, a scientific trend. The use of cloth material properties to design the automated hardware is natural and inevitable. Research should be continued into the correlations between cloth properties and machine parameters.

8 Conclusions

The foundations for a picker placement expert system have been developed. This expert system uses both scientific and empirical rules to determine the proper locations for cloth pickers so that good cloth control can be maintained. The user simply inputs the cloth geometry, cloth properties, and the edge to be moved, then the expert system determines a set of suitable picker locations for that edge.

In the process of formulating rules for the expert system, an elastica cloth model was developed to predict the behavior of a two dimensional elastica contacting a frictional surface. The model revealed a number of features of the behavior of cloth under these conditions.

1) There are three distinct modes of behavior.
   a) Complete slip
   b) First stick, then slip
   c) Complete stick

2) When the dimensionless parameter $\beta^{1/3}$ exceeds 2.95, the "first stick, then slip" mode does not occur.

The elastica cloth model leads to the development of one of the rules of the picker placement expert system, i.e. the picker-to-edge rule. This rule simply states that the length of unsupported cloth should not exceed the length at which sticking starts to occur. A second rule, the inter-

picker spacing rule, dictates how close pickers can be placed together. This rule is based on the cloth shear stiffness. These rules, along with a couple of simpler rules, were shown to be sufficient to determine suitable sets of picker locations. With these suitable sets of picker locations, the edge of a piece of cloth can be picked up, moved over, and put back down on a flat surface without distorting the cloth.

Future Work

If this thesis work were to be continued, the next step would be actual implementation of the expert system for picker placement. This would demonstrate how such a system could be used, both as a productivity tool and a design tool. In addition, the expert system could be expanded to handle most aspects of automated folding. These aspects would include folding trajectories, vacuum pattern, and creaser movement. An expert system which could aid in some or all of these areas would greatly increase the flexibility of the folding module. Work should continue to try to correlate cloth properties to machine parameters.

References

1 Abernathy, Fredrick H., and Don Pippins, $(TC)^2$ Apparel, Textile and Education at its Best., Bobbin Magazine, September 1986.

2 Bernardon, Edward, Robots in the Apparel Industry, October 1st, 1986, Charles Stark Draper Laboratory, Inc. Report number TCI/189-A.

3 Parker, J.K. et al, Robot Fabric Handling for Automating Garment Manufacturing, Journal of Engineering for Industry, February 1983, Vol. 105.

4 Ito, Ann Toshiko, Design of a Vacuum Picker For Automated Handling of Textiles, M.I.T. B.S Thesis in Mechanical Engineering, June 1987.

5 Murray, John M., Single-Ply Pick-up Devices, Apparel Research Journal, December 1975 Volume III No. III, p. 87-99.

6 Present State of Automated Sewing System Development, by Technology Research Association of Automated Sewing Systems

7 Lloyd, D.W. et al, Folding of Heavy Fabric Sheets, Univ. of Manchester Inst of Sci & Technol, Engl, International Journal of Mechanical Sciences v 20, n 8, 1978, p 521-527.

8 Wang, C.Y., Critical Review of the Heavy Elastica, Michigan State Univ, East Lansing, MI, International Journal of Mechanical Sciences, v.28, n.8, 1986, p 549-559

9 Crandall, Dahl, Lardner, An Introduction to the Mechanics of Solids, McGraw-Hill Book Company, 1972

10 Press, William H., Numerical Recipes, The Art of Scientific Computing; Cambridge University Press, 1986

Appendix A: Pants Assembly Instructions

The following is the instructions for manual assembly of men's dress pants (Simplicity 6668). These steps are included in this thesis for comparison with the automated assembly line. (Note: See Figures A-1, A-2, A-3, A-4, A-5, A-6, A-7, A-8, and A-9 following the text.)

Manual Pants Assembly

Front

1) Reinforce stitch the left pant front. Piece P.

2) N/A.

Yoke and Pockets

3) Attach front pocket facing (R) to front pocket (S).

4) Attach front pocket (R, S) to pants front (P).

5) Turn pocket facing inside, i.e. flip the pocket to the inside half of the pants.

6) Top stitch the front pocket edge.

7) Attach the yoke (T) to the yoke and pocket piece (U).

8) Sew the pocket halves together, i.e. R/S and T/U.

9) Stitch the yoke along the outseam, waist seam, and the fly seam

10) Bar tack the stress points.

11) Blanket stitch the pocket opening.

Front Flap

12) Sew front flap to interfacing, trim corners.

13) Turn flap inside out, top stitch edge.

14) Baste flap to pant front.

Left Fly

15) Attach left fly (G) to left pant front (P).

Zipper

16) Sew zipper onto left fly (G).

17) Turn left fly (G) in and baste on curved edge.

18) Top stitch left fly (G) to pants front (P).

Right Fly

19) Make a clip at the small dot on the right pant front, press under 3/8 inch (1 cm).

20) Baste side of zipper to right pants front (P).

21) Stitch lining (G) to fly section (G), clip curves.

22) Turn fly inside out, Baste fly section (G) to right pants front (P).

23) Top stitch fly section through zipper onto right pants front.

Back

24) Stay stitch upper edge of pant back (V), make dart.

Back Welt Buttonhole Pockets and Flap

25) To reinforce pocket opening stitch along pocket lines.

26) Fold welts (K) in half and press. Baste welts (K) onto pocket lines with the raw edges of the welts together.

27) Apply interfacing to back flap (J). Stitch and trim.

28) Turn back flap (J) inside out, top stitch edges.

29) Baste flap (J) to upper welt (K).

30) Press under 1/4 inch (6 mm) on back pocket facing (L).

31) Stitch back pocket facing (L) to back pocket (W) and press 1/4 inch (6 mm) along the long pocket edges and stitch.

32) Stitch back pocket (W) to pant back (V), cut pocket opening.

33) Turn pocket to inside of pants.

34) Sew edges of clipped corners.

35) Fold pocket and sew.

36) Fold top part of pants down from the waistline and sew top edge of pocket.

37) Baste upper edge of pants and pocket together.

38) Make button hole and sew on button.

Side Seams

39) Move pants back onto pants front and align the side edges, then sew the outseam.

Waistband

40) Attach interfacing to waistband (Y), trim to 1/4 inch (6 mm), turn and press edges.

41) Attach waistband to inside of pants and ease to fit.

42) Fold ends of waistband, Right sides together stitch and trim.

43) Turn band inside out on the ends, top stitch waist band to outside of pants.

Inner Leg Seams

44) Align inseam and sew

Center Seam (Crotch Seam)

45) Align and sew center seam.

Tailored Leg Hems

46) N/A.

47) Hem pant leg.

48) Cuff pant leg.

49) Tack pant cuff.

Finishing

50) Crease pant legs.

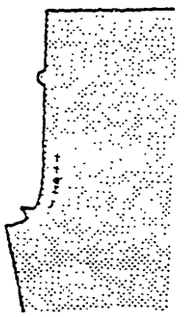
51) N/A.

52) Fold belt loops.

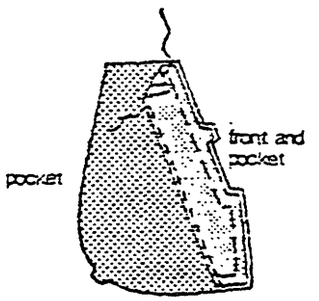
53) Stitch bottom of belt loops onto pants.

54) Stitch top of Belt loops onto pants

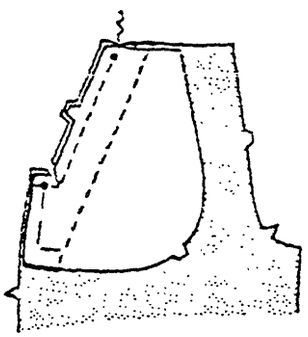
55) Make button and button hole for waist band.



1) Reinforce fly

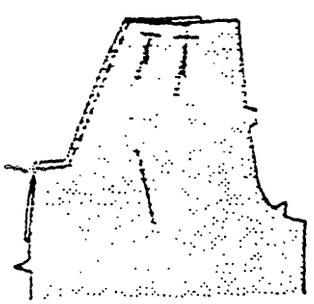


3) Sew pocket pieces

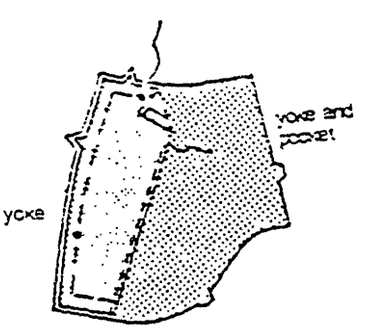


4) Sew pocket to front

5) Turn pocket to inside

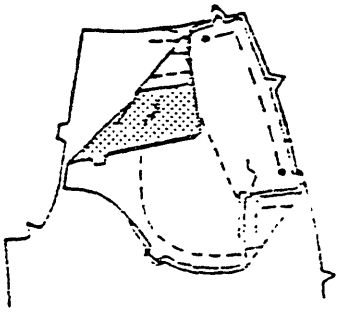


6) Top-stitch pocket edge

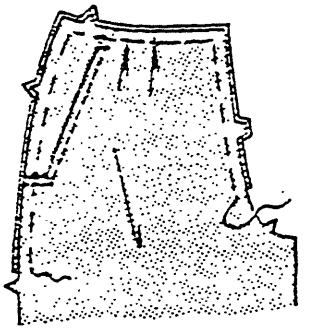


7) Sew pocket pieces

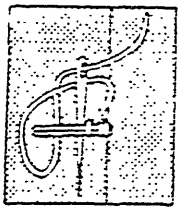
Figure A-1: Manual Assembly of Pants Steps 1 through 7.



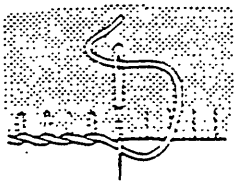
8) Sew Yoke to pocket



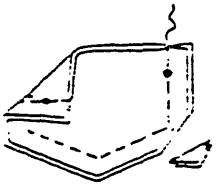
9) Sew Yoke to pants



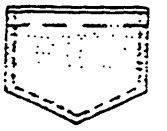
10) Bar Tack pocket corner



11) Blanket stitch pocket

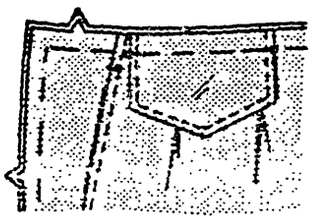


12) Sew front flap pieces

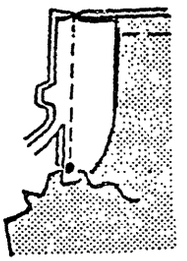


13) Turn and sew front flap

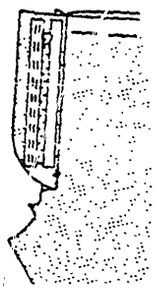
Figure A-2: Manual Assembly of Pants Steps 8 through 13.



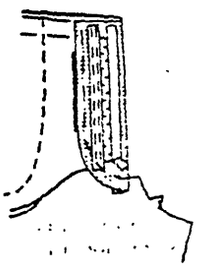
14) Sew flap to pants



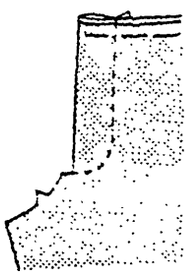
15) Sew fly to left front



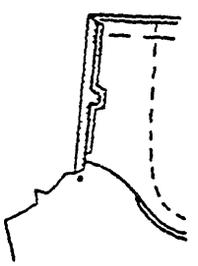
16) Sew zipper to fly



17) Turn fly inward

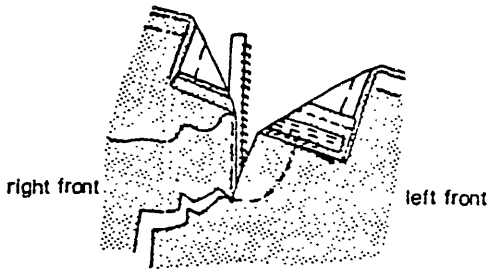


18) Top-stitch fly



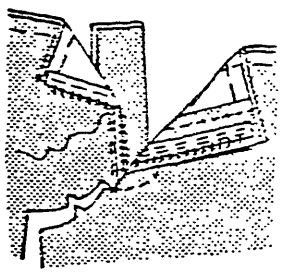
19) Press in right front

Figure A-3: Manual Assembly of Pants Steps 14 through 19.

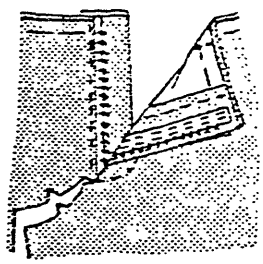


20) Sew zipper to right front

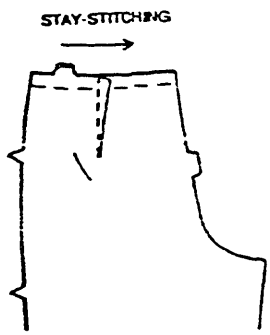
21) Sew fly sections



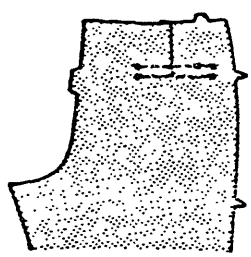
22) Baste fly to pants



23) Top stitch fly to pants

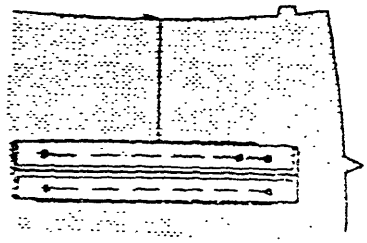


24) Dart and stitch back

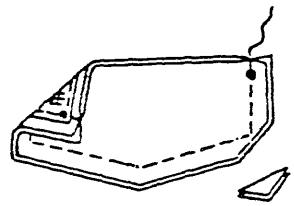


25) Sew pocket opening

Figure A-4: Manual Assembly of Pants Steps 20 through 25.

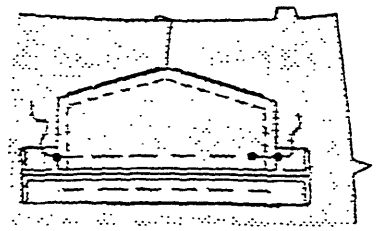


26) Sew welts to pants



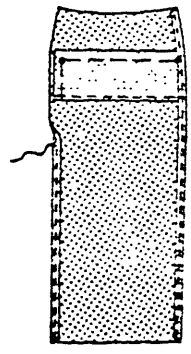
27) Sew back pocket pieces

28) Turn pocket and top-stitch



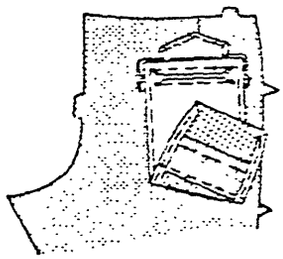
29) Sew pocket to welt

30) Hem back pocket facing.

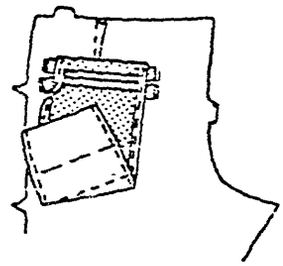


31) Sew pocket pieces

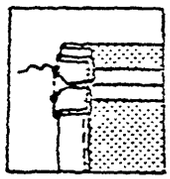
Figure A-5: Manual Assembly of Pants Steps 26 through 31.



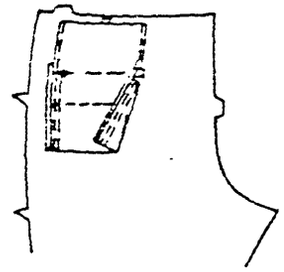
32) Sew back pocket to pants



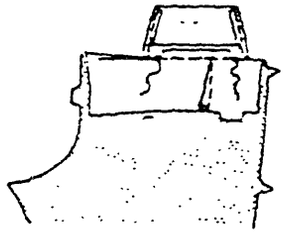
33) Turn pocket through pants



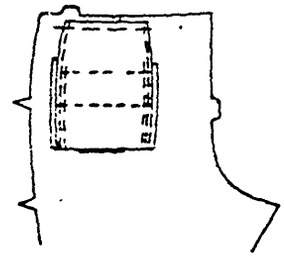
34) Sew welt corners



35) Sew pocket sides

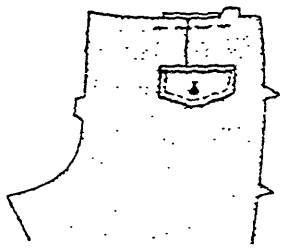


36) Sew pocket top

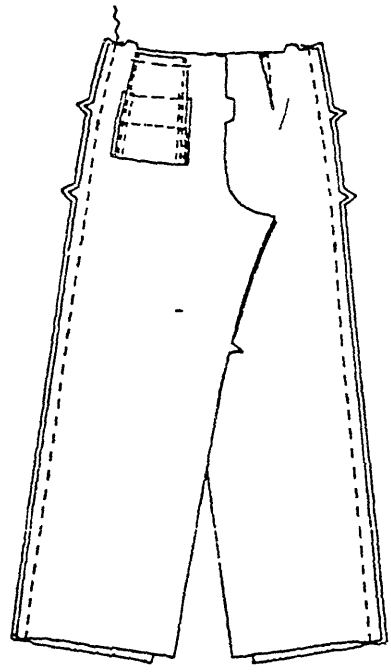


37) Sew upper edge of pocket

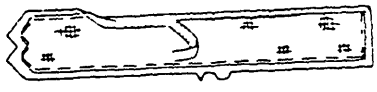
Figure A-6: Manual Assembly of Pants Steps 32 through 37.



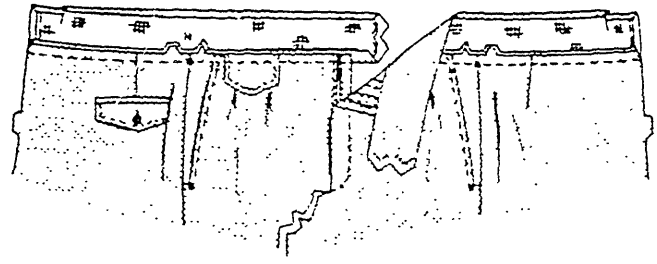
38) Button and button hole



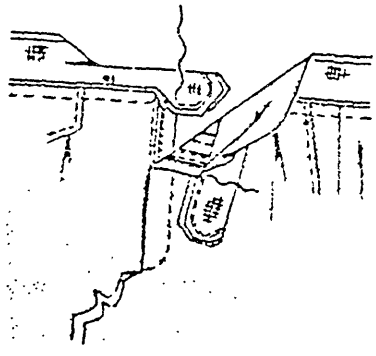
39) Side seams



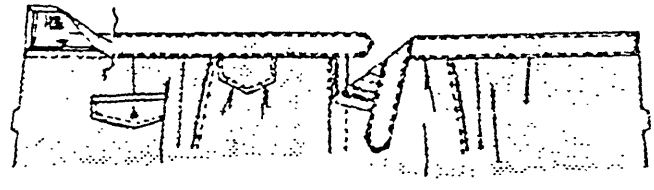
40) Sew interfacing and waistband



41) Sew waistband to inside

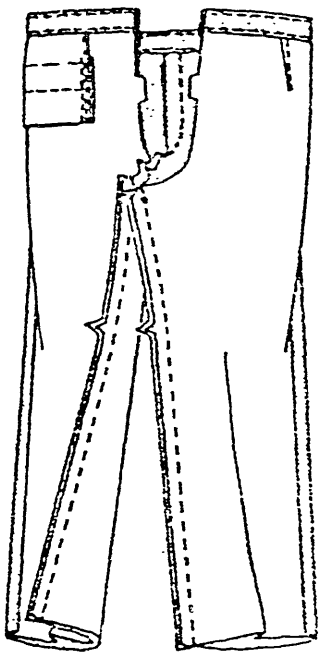


42) Sew ends of waistband

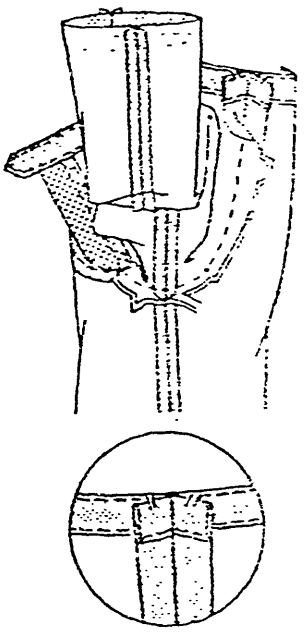


43) Sew waistband to outside

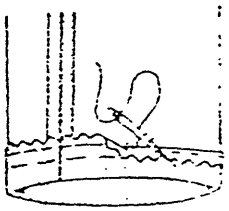
Figure A-7: Manual Assembly of Pants Steps 38 through 43.



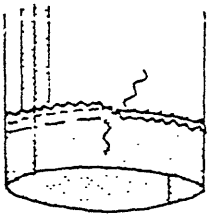
44) Sew inseam



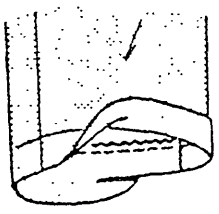
45) Sew crotch seam



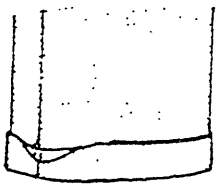
46) Hem pant leg



47) Hem pant leg

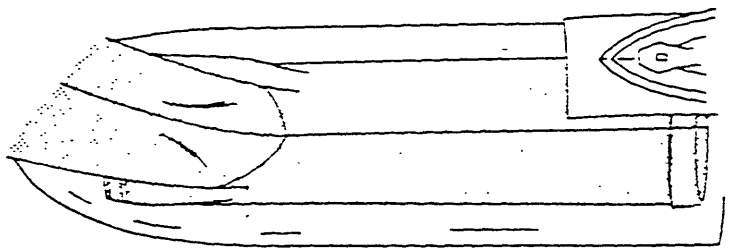


48) Cuff pant leg

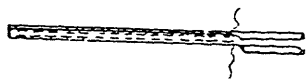


49) Tack pant leg

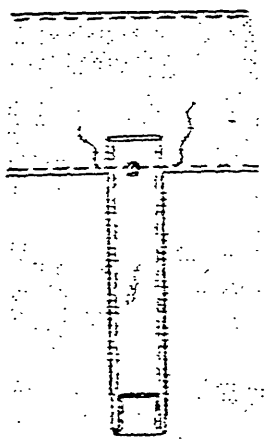
Figure A-8: Manual Assembly of Pants Steps 44 through 49.



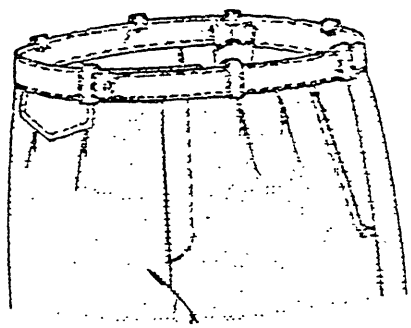
50) Crease pant leg



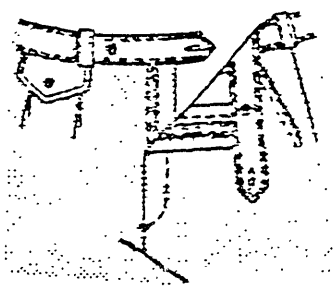
52) Make belt loops



53) Sew bottom of belt loops



54) Sew top of belt loops



55) Button and button hole

Figure A-9: Manual Assembly of Pants Steps 50 through 55.

The following is the procedures for automated pants assembly. Detailed procedures are given for the 10 folding/sewing stations and the manual sewing stations shown in Chapter 3. (Note: See Figures A-10, A-11, A-12, A-13a, A-13b, A-14a, A-14b, A-14c, A-15a, A-15b , A-16, A-17, A-18a, A-18b, A-19, and A-20 following the text.)

Steps for Automated Assembly of Pants

Folding/Sewing Station 1

3) Acquire front pocket (S) and front pocket facing (R) and place them on the folding table. Position front pocket facing (R) on front pocket (S). At sewing station sew the two parts together.

7) Acquire the Yoke (T) and the Yoke and Pocket (U). Position the Yoke (T) on top of the Yoke and Pocket (U) matching the outside edge. At a sewing station, sew the Yoke (T) to the Yoke and Pocket (U).

30) Acquire the back pocket facing (L) and fold under 1/4 inch (6 mm) along one long edge. This can be done at a sewing station using a hemming edge guide.

31) Acquire the back pocket (W). Position the back pocket facing (L) on the back pocket (W). At a sewing station, sew the facing (L) to the pocket (W). Hem 1/4 inch (6 mm) along the two long edges of the back pocket. This can be done at a sewing station using a hemming edge guide.

Folding/Sewing Station 2

4) Acquire pants front (P) and front pocket assembly (R/S). Turn pocket over and position on pants front (P). At sewing station sew pocket assembly (R/S) to pants front (P).

1) Acquire pants front (P) and place on the folding table. Send through sewing station and reinforce stitch near the fly.

Folding/Sewing Station 3

5) Fold pocket assembly (R/S) out so that the opposite side is facing up. Pickup the assembly along outseam and outer edge of pocket. Flip assembly over using positive flip out technique. Fold pocket back in on the wrong side of pant front (P). Use a compliant creaser to assure a flat fold. Because of the notch at the bottom of the pocket a second folding operation is necessary. Pickup the bottom edge of the pocket assembly and fold it up towards the waist. This should straighten out the notch in the pocket. Again the compliant creaser can be used to assure a flat fold.

6) At a sewing station, top stitch the pocket edging 1/4 inch (6 mm) away from the edge.

Folding/Sewing Station 4

8) Position the pants front (P/R/S) on top of the Yoke and Pocket assembly (U/T) such that the front pocket (R/S) can be folded out to match the Yoke and Pocket (U/T). Fold the front pocket (R/S) out to match up with the Yoke and Pocket (U/T) using the positive flip out technique. At a sewing station, sew the curved edge of the front pocket (R/S) to the Yoke and Pocket (U/T). Fold (R/S/U/T) on to piece the pants front (P). There will be a funny fold on piece (U/T) but that will come out in the next step.

Folding/Sewing Station 5

9) Pick up the pants front (P) along the inseam and position it on top of the Yoke (U/T) so that the yoke matches up with the fly. At a sewing station, sew the yoke to the pants front (P) along the fly seam, waist seam, and the outseam.

14) Position front flap on waist seam. Sew front flap to waist seam. Note this step can be combined with step 9.

15) Acquire fly (G) and position it (wrong side up) on top of the left pant front (P). At a sewing station, sew along the straight edge of the fly.

Folding/Sewing Station 6

16) Turn the fly (G) out so that the right side is facing out. Acquire the zipper and position it on the fly. At a sewing station, sew along the left edge of the zipper tape.

Folding/Sewing Station 7

24) Acquire pants back (V). Using a special machine, make the dart in the back of the pants near the waist seam. At a sewing station, sew along the waist seam.

Folding/Sewing Station 8

26) With a special machine, fold the welts (K) in half. Acquire the folded welts and position them on the outside of the pants back (V). At a sewing station, sew the welts on to the pants back (V) along the pocket lines.

29) Position the back flap (J/J) on the upper welt. At a sewing station, sew the flap (J/J) to the upper welt (W).

Folding/Sewing Station 9

39) Acquire pants back (V) and pants front (P). Position pant back (V) (wrong side up) on top of pants front (P) with outseam aligned. At a sewing station, sew the outseam.

Folding/Sewing Station 10

44) After step 39, align the inseam of (P) and (V). At a sewing station, sew the inseam of the pants.

Semi-Automatic and Manual Finishing Operations

Details

10) Bar tack points of stress using automatic bar tacking machine.

11) Blanket stitch work the pocket edges.

Finish Back Pocket

32) Sew back pocket (W/L) to pants back (V). Cut pocket and pants back to make pocket opening.

33) Turn pocket through pocket opening, i.e. to the inside of the pants.

34) Inside the pants back (V) fold over and sew the corners of the welts.

35) Fold the pocket in half and sew the long side seams of the pocket.

36) Fold the waist of the pants down so that there is a fold near the top pocket welt. Sew the top of the pocket to the inside of the pants.

37) Baste the upper edge of the pants and the pocket together at the waist.

38) On the outside of the pants press down the flap and sew on a button and a button hole.

Finish Zipper

17) Fold zipper under.

18) Top stitch fly in place.

19) Press under 3/8 inch (1 cm) along the right fly. Consult apparel designer about modifying this step.

20) Sew zipper in place on the right tape.

21) Position fly (G) on interfacing and sew.

22) Turn fly (G/G) inside out and attach to right pant front.

23) Top stitch fly into place.

Attach Waistband

41) Attach inside of waist band to pants.

43) Turn ends of waist band and sew waist band to the outside of the pants.

Crotch Seam

45) Sew the crotch seam.

Finish Pant Legs

47) On an automatic hemming machine hem the pant leg.

48) Cuff the pant leg.

49) Tack the pant cuff in place.

50) Crease the pant leg.

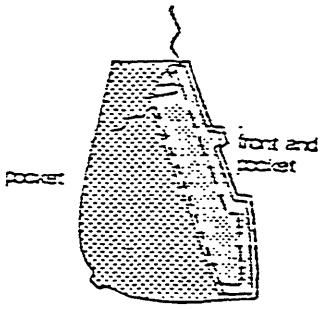
Belt Loops and Buttons

52) On a special machine, fold and sew the belt loops.

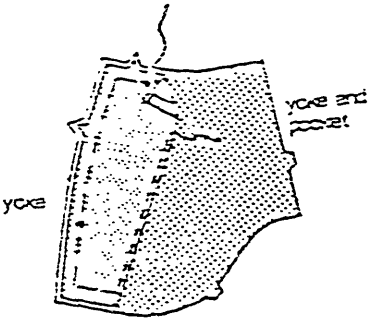
53) On a bar tack machine, tack the belt loops on to the bottom of the waist band.

54) On a bar-tack machine, tack the belt loop to the top of the waistband.

55) Make the button hole and sew the button on to the waistband.

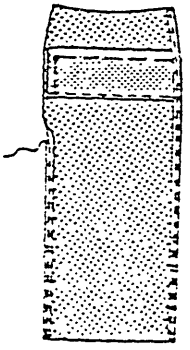


3) Sew pocket pieces



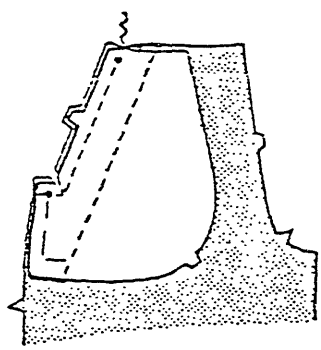
7) Sew pocket pieces

30) Hem back pocket facing.

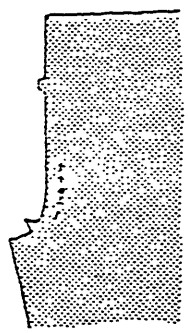


31) Sew pocket pieces

Figure A-10: Folding/Sewing Station 1, assembling pocket pieces.

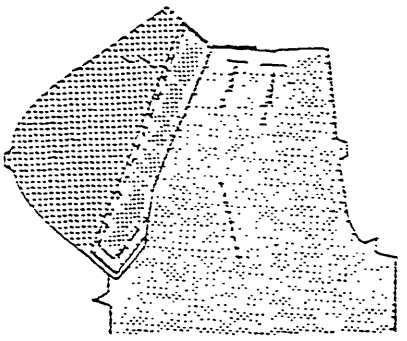


4) Sew pocket to front

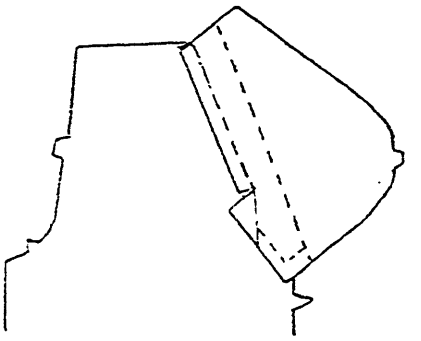


1) Reinforce fly

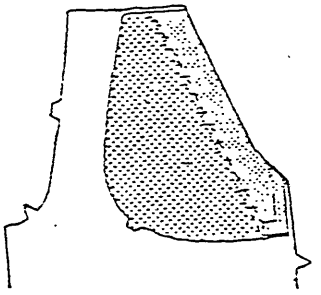
Figure A-11: Folding/Sewing Station 2, attaching front pocket to pants front.



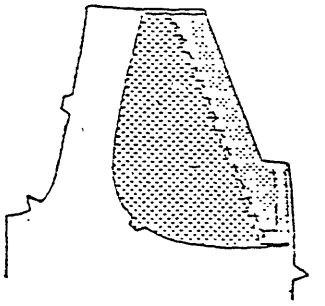
5a) Flip out pocket



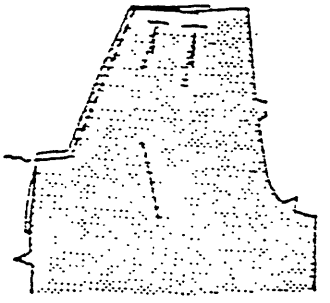
5b) Turn assembly over



5c) Flip pocket over

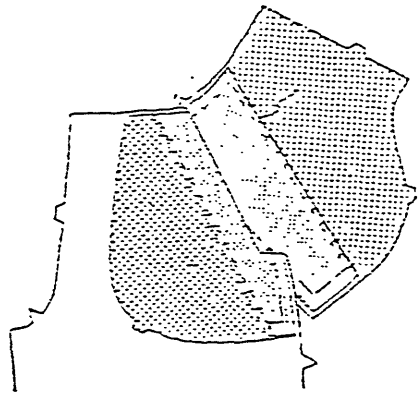


5d) Fix fold

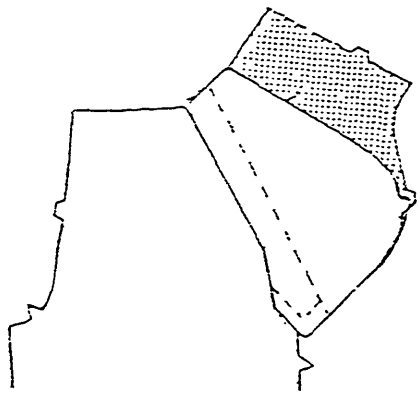


6) Top-stitch pocket edge

Figure A-12: Folding/Sewing Station 3, sewing front pocket edge.

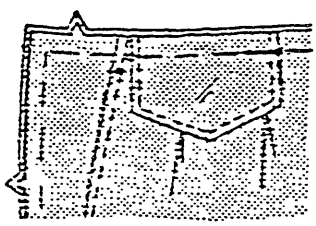


8a) Position pants on yoke

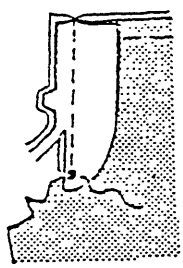


8b) Fold out pocket on yoke

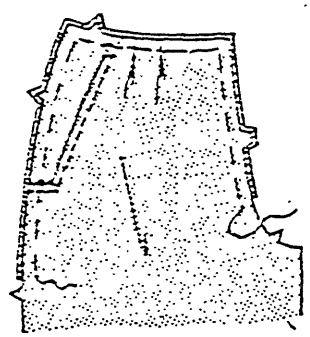
Figure A-13a: Folding/Sewing Station 4, attaching yoke to front pocket



14) Sew flap to pants

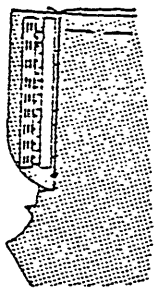


15) Sew fly to left front



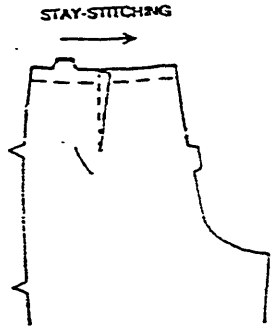
9) Sew Yoke to pants

Figure A-13b: Folding/Sewing Station 5, attaching yoke, fly, and flap



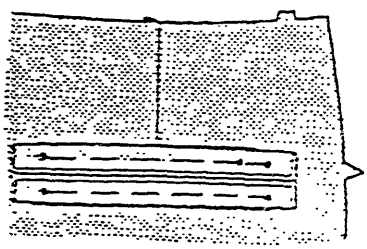
16) Sew zipper to fly

Figure A-14a: Folding/Sewing Station 6, attaching zipper to fiy

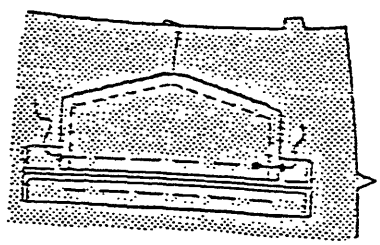


24) Dart and stitch back

Figure A-14b: Folding/Sewing Station 7, making dart in pants back.

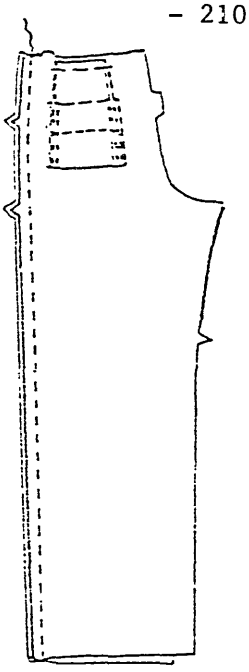


26) Sew welts to pants



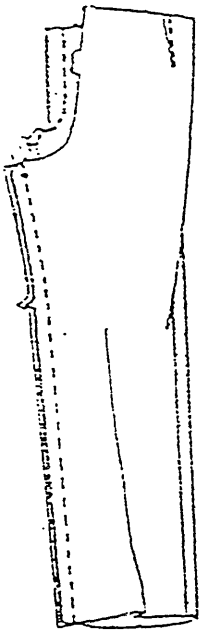
29) Sew pocket to welt

Figure A-14c: Folding/Sewing Station 8, starting back pocket.



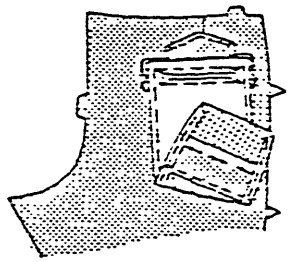
39) Side seams

Figure A-15a: Folding/Sewing Station 9, sewing outseam.

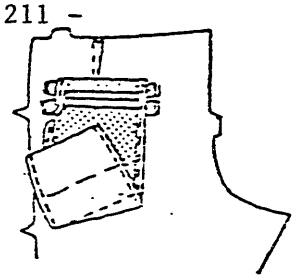


44) Sew inseam

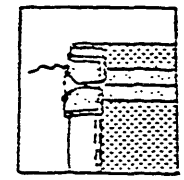
Figure A-15b: Folding/Sewing Station 10, sewing inseam.



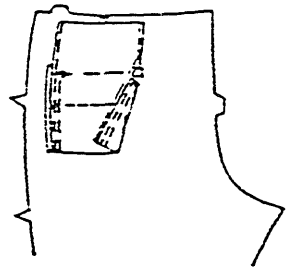
32) Sew back pocket to pants



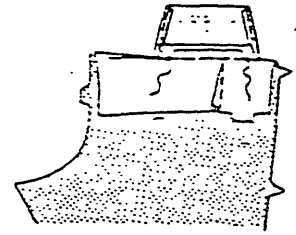
33) Turn pocket through pants



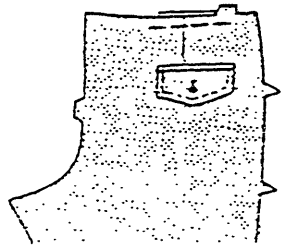
34) Sew welt corners



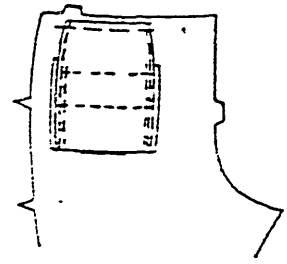
35) Sew pocket sides



36) Sew pocket top

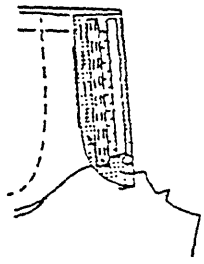


38) Button and button hole

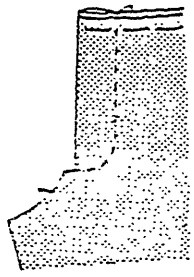


37) Sew upper edge of pocket

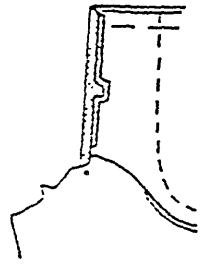
Figure A-16: Manual sewing, Finish back pocket.



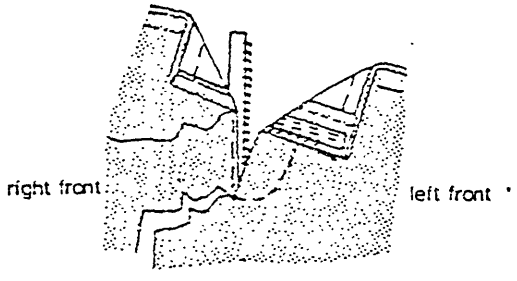
17) Turn fly inward



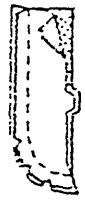
18) Top-stitch fly



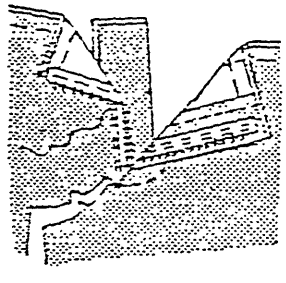
19) Press in right front



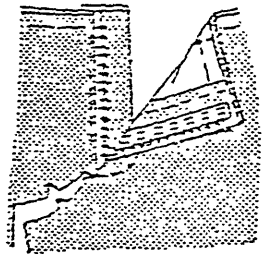
20) Sew zipper to right front



21) Sew fly sections

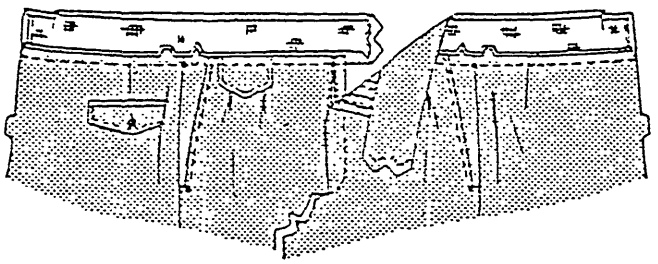


22) Baste fly to pants

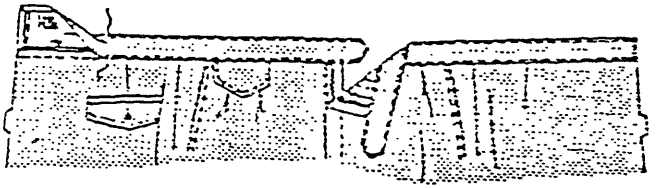


23) Top stitch fly to pants

Figure A-17: Manual sewing, Finish zipper.

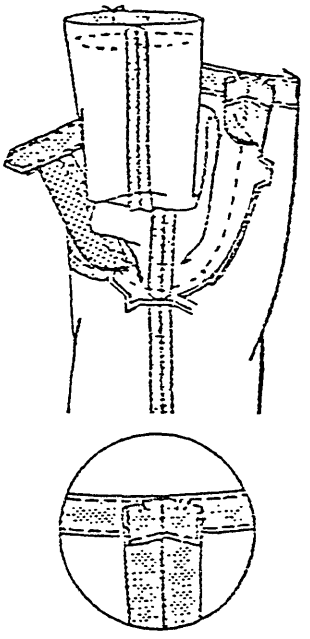


41) Sew waistband to inside



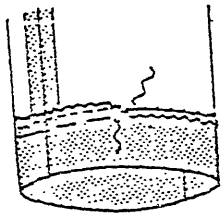
43) Sew waistband to outside

Figure A-18a: Manual sewing, Attach waistband.

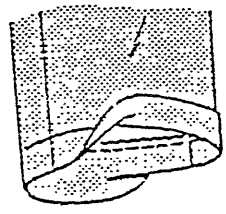


45) Sew crotch seam

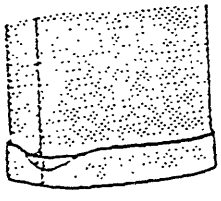
Figure A-18b: Manual sewing, Sew crotch seam.



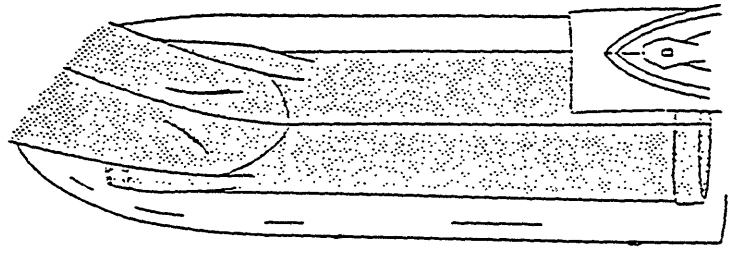
47) Hem pant leg



48) Cuff pant leg



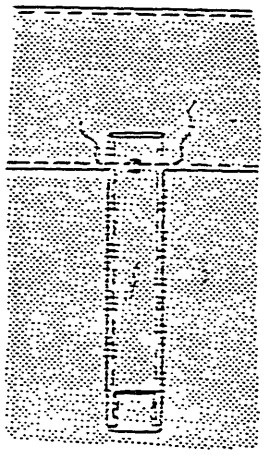
49) Tack pant leg



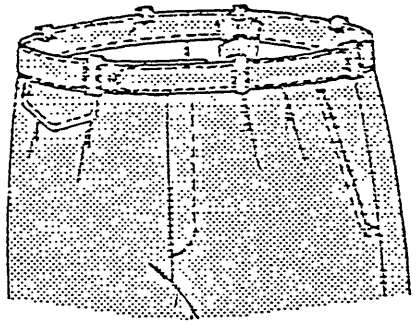
50) Crease pant leg

Figure A-19: Manual sewing, Hem pant leg.

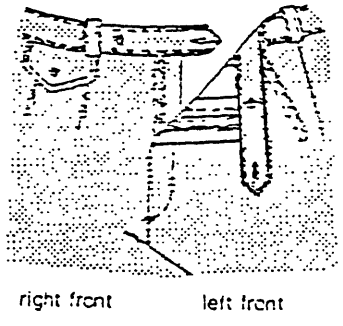
52) Make belt loops



53) Sew bottom of belt loops



54) Sew top of belt loops



55) Button and button hole

Figure A-20: Manual sewing, Attach belt loops.

Appendix B: Runge-Kutta Numerical Integration

In order to integrate the four ODE of the elastica (equations 4-4 through 4-7), the Runge-Kutta method has to be used four times. The equations are as follows, note that for simplicity equations (4-4) through (4-7) have been represented by the functions f(),g(),h() and i()

$$\begin{aligned} dF_x/ds &= f(s,F_x,F_y,M,\theta) \\ dF_y/ds &= g(s,F_x,F_y,M,\theta) \\ dM/ds &= h(s,F_x,F_y,M,\theta) \\ d\theta/ds &= i(s,F_x,F_y,M,\theta) \end{aligned} \qquad \text{(B-1)}$$

$$\begin{aligned} j_1 &= h\cdot f(s_0,F_{x0},F_{y0},M_0,\theta_0) \\ j_2 &= h\cdot f(s_0+\tfrac{1}{2}h,F_{x0}+\tfrac{1}{2}j_1,F_{y0}+\tfrac{1}{2}k_1,M_0+\tfrac{1}{2}l_1,\theta_0+\tfrac{1}{2}m_1) \\ j_3 &= h\cdot f(s_0+\tfrac{1}{2}h,F_{x0}+\tfrac{1}{2}j_2,F_{y0}+\tfrac{1}{2}k_2,M_0+\tfrac{1}{2}l_2,\theta_0+\tfrac{1}{2}m_2) \\ j_4 &= h\cdot f(s_0+h,F_{x0}+j_3,F_{y0}+k_3,M_0+l_3,\theta_0+m_3) \end{aligned} \qquad \text{(B-2)}$$

$$\begin{aligned} k_1 &= h\cdot g(s_0,F_{x0},F_{y0},M_0,\theta_0) \\ k_2 &= h\cdot g(s_0+\tfrac{1}{2}h,F_{x0}+\tfrac{1}{2}j_1,F_{y0}+\tfrac{1}{2}k_1,M_0+\tfrac{1}{2}l_1,\theta_0+\tfrac{1}{2}m_1) \\ k_3 &= h\cdot g(s_0+\tfrac{1}{2}h,F_{x0}+\tfrac{1}{2}j_2,F_{y0}+\tfrac{1}{2}k_2,M_0+\tfrac{1}{2}l_2,\theta_0+\tfrac{1}{2}m_2) \\ k_4 &= h\cdot g(s_0+h,F_{x0}+j_3,F_{y0}+k_3,M_0+l_3,\theta_0+m_3) \end{aligned} \qquad \text{(B-3)}$$

$$\begin{aligned} l_1 &= h\cdot h(s_0,F_{x0},F_{y0},M_0,\theta_0) \\ l_2 &= h\cdot h(s_0+\tfrac{1}{2}h,F_{x0}+\tfrac{1}{2}j_1,F_{y0}+\tfrac{1}{2}k_1,M_0+\tfrac{1}{2}l_1,\theta_0+\tfrac{1}{2}m_1) \\ l_3 &= h\cdot h(s_0+\tfrac{1}{2}h,F_{x0}+\tfrac{1}{2}j_2,F_{y0}+\tfrac{1}{2}k_2,M_0+\tfrac{1}{2}l_2,\theta_0+\tfrac{1}{2}m_2) \\ l_4 &= h\cdot h(s_0+h,F_{x0}+j_3,F_{y0}+k_3,M_0+l_3,\theta_0+m_3) \end{aligned} \qquad \text{(B-4)}$$

$$
\begin{aligned}
m_1 &= h \cdot i(s_0, F_{x0}, F_{y0}, M_0, \theta_0) \\
m_2 &= h \cdot i(s_0 + \tfrac{1}{2}h, F_{x0} + \tfrac{1}{2}j_1, F_{y0} + \tfrac{1}{2}k_1, M_0 + \tfrac{1}{2}l_1, \theta_0 + \tfrac{1}{2}m_1) \\
m_3 &= h \cdot i(s_0 + \tfrac{1}{2}h, F_{x0} + \tfrac{1}{2}j_2, F_{y0} + \tfrac{1}{2}k_2, M_0 + \tfrac{1}{2}l_2, \theta_0 + \tfrac{1}{2}m_2) \\
m_4 &= h \cdot i(s_0 + h, F_{x0} + j_3, F_{y0} + k_3, M_0 + l_3, \theta_0 + m_3)
\end{aligned}
\qquad \text{(B-5)}
$$

$$
\begin{aligned}
F_{x1} &= F_{x0} + (j_1 + 2 \cdot j_2 + 2 \cdot j_3 + j_4)/6 \\
F_{y1} &= F_{y0} + (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)/6 \\
M_1 &= M_0 + (l_1 + 2 \cdot l_2 + 2 \cdot l_3 + l_4)/6 \\
\theta_1 &= \theta_0 + (m_1 + 2 \cdot m_2 + 2 \cdot m_3 + m_4)/6
\end{aligned}
\qquad \text{(B-6)}
$$

The subscripts 1, 2, 3, and 4 on variables j, k, l, and m are used to identify the four parts of the Runge-Kutta integration. The subscripts 0 and 1 on variables F_x, F_y, M, and θ are used to indicated the node points. These subscripts would continue on up to the "nth" node where n is the total number of nodes. The variable h is the step size in the independent variable s.

Appendix C: Computer Code for Elastica Model

The following pages contain the computer code for the non-linear elastica program. The program is written in "C" language and compiled by Computer Innovations C compiler version 2.3. For graphics, Media Cybernetics Halo subroutines were used.

The programs main menu is as follows

```
                Main Menu

  1   Load beam information file
  2   Save current beam information
  3   Edit beam information
  4   Run beam bending program
  5   Exit to DOS

  Enter choice:
```

If option 3 is selected, the user is presented with a second menu.

```
                    Editing cloth data

                      Filename: default.cth
             Beam length, inch: 2.000000
              Beam width, inch: 1.000000
    Cloth weight, lbm/in^2: 0.000286
Cloth stiffness, EI,lbf*in^2: 7.700E-005
    Coefficient of Friction: 0.800
    Number of Beam segments: 35
  Move down each step, inch: 0.002000
```

The user fills in the appropriated information and selects option 4 of the main menu to see elastica simulation.

```
/*                                                              */
/*    ELASTICA BEAM PROGRAM, BEAM.EXE                           */
/*                                                              */

#include "stdio.h"                      /* input/output functions */
#include "math.h"                       /* math functions */
#define MAX_NODES 151                   /* maximum number of nodes */
#define IEEE 1                          /* Halo, IEEE short real format */
#define NMAX 5                          /* max # of boundary conditions */


main()
{
    extern plot_beam();                 /* beam plotting function */
    extern int sgn();                   /* calculate the sign of a double */
    extern crt_srcp();                  /* set cursur position function */
    extern crt_mode();                  /* set graphics mode */
    extern crt_cls();                   /* clear screen function */
    extern double f_abs();              /* absolute value of a double */
    extern double m_guess();            /* guess of m[0] */
    extern double finput();             /* function from "jcbinput.obj" */
    extern int iinput();                /* function from "jcbinput.obj" */
    extern sinput();                    /* function from "jcbinput.obj" */
    extern int menu_logic();            /* function for menu control */
    extern v_vec();                     /* assemble free b.c. */
    extern f_vec();                     /* assemble descrepency vector  */
    extern FILE *fopen();               /* file opening function */
    extern int fclose();                /* file closing function */
    FILE *ptr;                          /* stream for file */
    FILE *ptr2;                         /* stream for file */
    char *name = "default.cth\0
                                            ";
                                        /* file name w/ cloth properties */
    char *name2 = "\0                 ";
    char *quest1 ="n\0  ";              /* question Y/N */
    char *quest2 ="n\0  ";              /* question Y/N */
    char *device = "haloibme.dev";      /* name of halo screen driver */
    char *pdevice = "haloepsn.prn";     /* name of halo print driver */
                                        /* Units are
                                        length = inches
                                        force  = pounds(lbf)
                                        time   = seconds
                                        angles = radians */

    double      fx[MAX_NODES];          /* X direction forces at nodes */
    double      fy[MAX_NODES];          /* Y direction forces at nodes */
    double       m[MAX_NODES];          /* Moments at the nodes */
    double       x[MAX_NODES];          /* X location of the nodes */
    double       y[MAX_NODES];          /* Y location of the nodes */
```

```
double       th[MAX_NODES];          /* angle at node */
double     dths[MAX_NODES];          /* dθ/ds at node */
double  x_erase[MAX_NODES];          /* data for erasing beam */
double  y_erase[MAX_NODES];          /* data for erasing beam */
double x2_desired[NMAX];             /* desired condition at x2 */
double v[NMAX];                      /* free boundary conditions at x1 */
double f[NMAX];                      /* discrepency vector at x2 */
double dv[NMAX];                     /* temp. discrepency vector */
double delv[NMAX];                   /* change in v for Jacobian */
double dfdv[NMAX][NMAX];             /* Jacobian for Newton Ralphson */
double save;                         /* temporary variable */
double m_max;                        /* Moment for horizontal beam */
double bend;                         /* bending length */
double cof = 0.8;                    /* Coefficient of Friction */
double l = 2.0;                      /* length of the beam in inches */
double b = 1;                        /* width of the beam in inches */
double h = 0.013;                    /* thickness of beam in inches */
double w = 0.000286;                 /* weight of the beam in lbm/in» */
double y_end_actual = -1.7;          /* actual end of beam */
double xmin,xmax,ymin,ymax;          /* min, max screen scale, inch */
double ei = 7.7e-5;                  /* flexural rigidity of beam */
double wt;                           /* total beam weight lbf */
double ls;                           /* segment length in inches */
double count=0.0;                    /* time of calculation */
double x_target = 0.0;               /* desired location of end */
double y1=0.0,y2=0.0,y3=0.0;         /* last 3 values of y_end */
double tol1 = 1.0;                   /* tolerance on y_end calculation */
double clip = 0.50;                  /* percent clip limit on changes */
double mmove = 0.002;                /* the step size for moving cloth */
double amove = 0.005;                /* the keyboard step size,moving cloth */
double limit = 0.0;                  /* location of the table */
float xmint,xmaxt,ymint,ymaxt;       /* temporary */
unsigned int d_key,d_key_l;          /* keyboard input variables */
unsigned int d_key_h;                /* keyboard input variables */
int nbc = 3;                         /* number of boundary conditions */
int indx[NMAX];                      /* Row pertubation in LU decomp. */
int n = 35;                          /* number of beam segments */
int j,k,q,q1,q2;                     /* general index & looping  */
int arrow;                           /* flag for IBM arrow keys */
int print_1 = 0;                     /* flag for printing data */
int print_2 = 0;                     /* flag for printing data */
int iteml,item2;                     /* flag for position , input mode */
int max_item2 = 10;                  /* total number of menu items */
int junk;                            /* self-explainitory variable */
int gmode = 4;                       /* graphics mode 4 */
int gcolor = 8;                      /* graphics color */
int b_attr = 2;                      /* border character attribute */
int t_attr = 3;                      /* text character attribute */
int error_attr = 4;                  /* error character attribute */
int format = IEEE;                   /* floating point format */
```

```
    int print_menu = 1;                 /* flag for displaying menu */
    int cond = 0;                       /* 1 = beam on table */
    int cong = 0;                       /* 0 has not converged */
    float x1t,y1t,x2t,y2t;              /* temporary floats */

    /*                                                                  */
    /*  Check for the presence of Halo device drivers                   */
    /*                                                                  */
   crt_mode(3);                         /* get out of graphic mode */

   ptr = fopen(device, "r");            /* check for Halo screen driver */
   if(! ptr)abort(" %s not found\n",device);
   fclose(ptr);

   ptr = fopen(pdevice, "r");           /* check for Halo print driver */
   if(! ptr)abort(" %s not found\n",pdevice);
   fclose(ptr);


/*                                                                      */
/*  Present the main menu                                               */
/*                                                                      */

/* The options in the menu below are selected by a switch/case loop */
/* Item number 0 is the main menu itself */


item1 = 0;                              /* goto main menu */
while(item1 != -1)                      /* while loop for the switch */
switch(item1) {                         /* beginning of switch */


case 0:                                 /* Main Menu case */

if(print_menu == 1){                    /* does menu need to be printed */
  crt_cls();

  crt_srcp(5,21,0);                     /* print out box */
  printc("┌──────────────────────────────────────┐",b_attr);
  crt_srcp(6,21,0);
  printcv("||||||||||",b_attr);
  crt_srcp(6,58,0);
  printcv("||||||||||",b_attr);
  crt_srcp(17,21,0);
  printc("└──────────────────────────────────────┘",b_attr);
```

```
  crt_srcp(6,22,0);                        /* print out text of menu */
  printc("              Main Menu                ",t_attr);
  crt_srcp(7,22,0);
  printc("                                       ",t_attr);
  crt_srcp(8,22,0);
  printc("  1   Load beam information file       ",t_attr);
  crt_srcp(9,22,0);
  printc("  2   Save current beam information    ",t_attr);
  crt_srcp(10,22,0);
  printc("  3   Edit beam information            ",t_attr);
  crt_srcp(11,22,0);
  printc("  4   Run beam bending program         ",t_attr);
  crt_srcp(12,22,0);
  printc("  5   Exit to DOS                      ",t_attr);
  }

     while(item1<1 || item1>5){
       crt_srcp(14,24,0);
       printf("Enter choice:                  ");
       crt_srcp(14,38,0);
       item1 = iinput(14,38,40,item1,&arrow);
       crt_srcp(15,22,0);
       printf("                                      ");
       crt_srcp(16,22,0);
       printf("                                      ");
       }
  break;                                   /* end of case 0 */
```

```
/*                                                                    */
/*   ┌──────────────────────────────────────────────────────────┐     */
/*   │                Load a beam information file              │     */
/*   └──────────────────────────────────────────────────────────┘     */
```

```
case 1:                                    /* Get info from a file, case 1 */

  crt_srcp(14,24,0);
  printf("Enter name of file to LOAD     ");
  crt_srcp(15,24,0);
  printf("%s",name);                       /* print name of file in memory */
  sinput(15,24,57,name,&arrow);            /* User to enter file name */
  crt_srcp(15,24,0);
  printf("%s",name);                       /* Echo back name */


  ptr = fopen(name,"r");                   /* read data from file */
  if(ptr) {                                /* Valid stream ?*/
    fscanf(ptr,"%le %le %le %le %le %d",&l,&b,&h,&w,&ei,&n);
    fclose(ptr);
    crt_srcp(16,24,0);
```

```
    printc("File LOADed",t_attr);
    }
  else {                                  /* not a valid stream */
    crt_srcp(16,24,0);
    printc("Error could not open file\007",error_attr);
    }

    item1 = 0;                            /* return to Main Menu */
    print_menu = 0;                       /* don't reprint menu */
    break;                                /* end of case 1 */


/*  ┌──────────────────────────────────────────────────────┐  */
/*  │          Save current beam information to a file     │  */
/*  └──────────────────────────────────────────────────────┘  */

case 2:                                   /* Saving current beam info, case 2 */

  crt_srcp(14,24,0);
  printf("Enter name of file be to SAVEd ");
  crt_srcp(15,24,0);
  printf("%s",name);                      /* print name of file in memory */
  sinput(15,24,57,name,&arrow);           /* User to enter file name */
  crt_srcp(15,24,0);
  printf("%s",name);                      /* Echo back name */

        ptr = fopen(name,"w");            /* write file to disk */
        if(ptr) {                         /* Valid stream ? */
          fprintf(ptr,"%le %le %le %le ",l,b,h,w);
          fprintf(ptr,"%le %d",ei,n);
          fclose(ptr);
          crt_srcp(16,24,0);
          printc("File SAVEd",t_attr);
          }
        else {                            /* not a valid stream */
          crt_srcp(16,24,0);
          printc("Error could not open file\007",error_attr);
          }

  item1 = 0;                              /* return to Main Menu */
  print_menu = 0;                         /* don't reprint menu */
  break;                                  /* end of case 2 */


/*  ┌──────────────────────────────────────────────────────┐  */
/*  │              Edit current beam information           │  */
/*  └──────────────────────────────────────────────────────┘  */

case 3:                                   /* Editting file info, case 3 */
```

```
item2 = 0;                              /* default to first menu question */
crt_cls();                              /* Clear the screen */
      /* Print the screen templete*/
      /* This part of the text will not change */
      crt_srcp(4,27,0);
      printf("Editting cloth data");
      crt_srcp(6,10,0);
      printf("                    Filename: %s",name);
      crt_srcp(7,10,0);
      printf("          Beam length, inch: %f",l);
      crt_srcp(8,10,0);
      printf("           Beam width, inch: %f",b);
      crt_srcp(9,10,0);
      printf("      Cloth weight, lbm/in»: %f",w);
      crt_srcp(10,10,0);
      printf("Cloth stiffness, EI,lbf*in»: %.3e",ei);
      crt_srcp(11,10,0);
      printf("   Coefficient of Friction : %.3f",cof);
      crt_srcp(12,10,0);
      printf("    Number of beam segments: %d",n);
      crt_srcp(13,10,0);
      printf("  Move down each step, inch: %f",mmove);
      crt_srcp(16,10,0);
      printf("    Dim.less Bending length: %f",(w * b * pow(l,3.0))/ei);
      crt_srcp(17,10,0);
      printf("       Clip limit on change: %f",clip);
      crt_srcp(24,10,0);
      printf("Are you finished editting Y/N ? %s",quest2);


while(item2 != -1){                     /* begin of "item2" while */

    if(item2 == 0){                     /* Enter file name */
      sinput(6,39,60,name,&arrow);
      crt_srcp(6,39,0);
      printf("%s",name);
      item2 = menu_logic(item2,max_item2,arrow);
    }

    if(item2 == 1){                     /* Enter beam length */
      l = finput(7,39,60,1,&arrow);
      crt_srcp(7,39,0);
      printf("%f",l);
      item2 = menu_logic(item2,max_item2,arrow);
      }

    if(item2 == 2){                     /* Enter beam width */
      b = finput(8,39,60,b,&arrow);
      crt_srcp(8,39,0);
      printf("%f",b);
```

```
   item2 = menu_logic(item2,max_item2,arrow);
   }

 if(item2 == 3){                    /* Enter cloth weight */
   w = finput(9,39,60,w,&arrow);
   crt_srcp(9,39,0);
   printf("%f",w);
   item2 = menu_logic(item2,max_item2,arrow);
   }

 if(item2 == 4){                    /* Enter cloth stiffness */
   ei = finput(10,39,60,ei,&arrow);
   crt_srcp(10,39,0);
   printf("%.3e",ei);
   item2 = menu_logic(item2,max_item2,arrow);
   }

 if(item2 == 5){                    /* Enter guess of cloth stiffness */
   cof = finput(11,39,60,cof,&arrow);
   crt_srcp(11,39,0);
   printf("%.3f",cof);
   item2 = menu_logic(item2,max_item2,arrow);
   }


 if(item2 == 6){                    /* Enter the number of beam segments */
   n = iinput(12,39,60,n,&arrow);
   crt_srcp(12,39,0);
   printf("%d",n);
   item2 = menu_logic(item2,max_item2,arrow);
   }

 if(item2 == 7){                    /* actual y vertical deflection */
   mmove = finput(13,39,60,mmove,&arrow);
   crt_srcp(13,39,0);
   printf("%f",mmove);
   item2 = menu_logic(item2,max_item2,arrow);
   }

 if(item2 == 8){                    /* display bending length */
   crt_srcp(16,39,0);
   printf("%f",(w * b * pow(1,3.0))/ei);
   item2 = menu_logic(item2,max_item2,arrow);
   }


 if(item2 == 9){                    /* Enter iteration tolerance */
   clip = finput(17,39,60,clip,&arrow);
   crt_srcp(17,39,0);
   printf("%f",clip);
```

```
        item2 = menu_logic(item2,max_item2,arrow);
        }


      if(item2 == 10){                  /* Exit editting routine */
        sinput(24,42,43,quest2,&arrow);
        crt_srcp(24,42,0);
        printf("%s",quest2);
        item2 = menu_logic(item2,max_item2,arrow);
        if(quest2[0] == 'y' || quest2[0] == 'Y'){
           item2 = -1;
           print_menu = 1;
           item1 = 0;
           }
      }

   }                                    /* end of "item2" while */

   break;                               /* end of case 3 */


/*                                                                    */
/*  Solve the Bernoulli-Euler Beam equation                           */
/*                                                                    */

case 4:                                 /* case 4, do beam calculation */
   /* Do some graphics stuff first  */
   setieee(&format);                    /* set floating point format */
   setdev(device);                      /* set graphics driver */
   setprn(pdevice);                     /* set HALO print driver */
   initgraphics(&gmode);                /* open graphics environment */
   ymin = -1/20.0-1;                    /* Set up Y axis scaling */
   ymax = 1/20.0;                       /* slightly longer than beam length */
   xmin = -(ymax-ymin)*1.3333*.05;      /* Set up X axis scaling */
   xmax =  (ymax-ymin)*1.3333*.95;      /* screen ratio of 4/3 */

   x1t = xmin;
   x2t = xmax;
   y1t = ymin;
   y2t = ymax;
   setworld(&x1t,&y1t,&x2t,&y2t);       /* set up the coordinate system */
   setwindow(&x1t,&y1t,&x2t,&y2t);


/*                                                                    */
/*  Calculate some beam properties                                    */
/*                                                                    */

wt = b*l*w;                             /* total beam weight lbf */
```

```
ls = l/n;                                   /* segment length in inches */
m_max = -0.5 * w * b * ( l * l);            /* moment for horiz. beam */
bend = (w * b * pow(l,3.0))/ei;             /* bending length */
crt_srcp(2,60,0);
 printf("Hit ESC to end");
crt_srcp(16,60,0);
 printf("c = %3.3f",bend);

/*                                                                      */
/*   Calculate the initial position of each node                        */
/*                                                                      */

    for (j = 0; j <= n; ++j) {
        x_erase[j] = 0.0;
        y_erase[j] = 0.0;
        m[j] = 0.0;
        fx[j] = 0.0;
        fy[j] = 0.0;
        x[j] = 0.0;
        y[j] = 0.0;
        th[j] = 0.0;
        }
        count = 0.0;
        y1 = 1.0;  y2 = 2.0; y3 = 3.0;
        limit = 0.0;
     /*                                                                 */
     /*   Set Boundary Conditions on one end of the Elastica            */
     /*                                                                 */

      x[0] = 0;                             /* fixed boundary condition */
      y[0] = 0;                             /* fixed boundary condition */
      th[0] = 0;                            /* fixed boundary condition */
      fx[0] = 0;                            /* guessed boundary condition */
      fy[0] = - w * 1* b;                   /* guessed boundary condition */
      m[0] = m_guess(bend)*m_max;           /* guessed boundary condition */
      x2_desired[1] = 0;                    /* fx at point x2, fixed b.c. */
      x2_desired[2] = 0;                    /* fy at point x2, fixed b.c. */
      x2_desired[3] = 0;                    /*  m at point x2, fixed b.c. */
      delv[1] = 0.0000001;                  /* variation of fx[0] */
      delv[2] = 0.0000005 * fy[0];          /* variation of fy[0] */
      delv[3] = 0.0000005 * m[0];           /* variation of  mx[0] */


/*                                                                      */
/*   Start Beam calculation                                             */
/*                                                                      */
 cond = 0;                                  /* beam is not contacting table */
 j = 1;                                     /* init condition for while loop */
   while(j) {                               /* loop for Beam calculation */
```

```
count = count + 1;

/*  ┌──────────────────────────────────────────────────────┐  */
/*  │                Trial Integration of Equations        │  */
/*  └──────────────────────────────────────────────────────┘  */

v_vec(fx[0],fy[0],m[0],v);
rungekutta(fx,fy,m,th,x,y,ei,ls,w,b,n);
f_vec(fx[n],fy[n],m[n],x[n],y[n],cond,f,x2_desired);

/*  ┌──────────────────────────────────────────────────────┐  */
/*  │                  Evaluate Jacobian                   │  */
/*  └──────────────────────────────────────────────────────┘  */

save = fx[0];                       /* VARY fx[0] */
fx[0] = fx[0] + delv[1];
  v_vec(fx[0],fy[0],m[0],v);
  rungekutta(fx,fy,m,th,x,y,ei,ls,w,b,n);
  f_vec(fx[n],fy[n],m[n],x[n],y[n],cond,dv,x2_desired);
  for(q=1; q<=nbc; q++)
     dfdv[q][1] = (dv[q] - f[q])/delv[1];
fx[0] = save;


save = fy[0];                       /* VARY fy[0] */
fy[0] = fy[0] + delv[2];
  v_vec(fx[0],fy[0],m[0],v);
  rungekutta(fx,fy,m,th,x,y,ei,ls,w,b,n);
  f_vec(fx[n],fy[n],m[n],x[n],y[n],cond,dv,x2_desired);
  for(q=1; q<=nbc; q++)
     dfdv[q][2] = (dv[q] - f[q])/delv[2];
fy[0] = save;


save = m[0];                        /* VARY m[0] */
m[0] = m[0] + delv[3];
  v_vec(fx[0],fy[0],m[0],v);
  rungekutta(fx,fy,m,th,x,y,ei,ls,w,b,n);
  f_vec(fx[n],fy[n],m[n],x[n],y[n],cond,dv,x2_desired);
  for(q=1; q<=nbc; q++)
     dfdv[q][3] = (dv[q] - f[q])/delv[3];
m[0] = save;


 for(q=1;q<=nbc;++q)                /* {δV} = inv[α]·{-F} */
    dv[q] = -f[q];

ludcmp(dfdv,nbc,NMAX,indx,&save); /* LU decomposition of Jacobian */
```

```
    lubksb(dfdv,nbc,NMAX,indx,dv);  /* solving equations */

                                    /* Clip limit the changes in V[]  */
       if(f_abs(dv[1]) > (clip * f_abs(v[1])) ||
          f_abs(dv[2]) > (clip * f_abs(v[2])) ||
          f_abs(dv[3]) > (clip * f_abs(v[3])) )
            for(q=1;q<=nbc;++q)
                 {
                 dv[q] = clip * dv[q];
                 crt_srcp(9,60,0);
                 printf("clipping %d",q);
                 /* limit changes to clip% of current value */
                 }
       else
          {                         /* erase */
          crt_srcp(9,60,0);
          printf("           ");
          }

    for(q=1;q<=nbc;++q)             /* increment boundary parameters  */
        v[q] = v[q] + dv[q];


    fx[0] = v[1];                   /* Try corrected guesses of */
    fy[0] = v[2];                   /* boundary parameters. */
     m[0] = v[3];
     v_vec(fx[0],fy[0],m[0],v);
     rungekutta(fx,fy,m,th,x,y,ei,ls,w,b,n);
     f_vec(fx[n],fy[n],m[n],x[n],y[n],cond,f,x2_desired);


/*  ┌──────────────────────────────────────────┐  */
/*  │          Plot the shape of the beam      │  */
/*  └──────────────────────────────────────────┘  */

plot_beam(n,l,x,y,x_erase,y_erase);

for(q=0; q<=n; q++)                 /* back up data */
   {
   x_erase[q] = x[q];
   y_erase[q] = y[q];
   }

/*  ┌──────────────────────────────────────────┐  */
/*  │      What is the condition of the beam?  │  */
/*  └──────────────────────────────────────────┘  */
crt_srcp(17,60,0);
printf("             ");

if(limit == 0.0)
```

```
    cond = 0;                          /* not contacting */
else
    if(f_abs(fx[n]) < 0.995 * cof * f_abs(fy[n]))
        cond = 1;                      /* contacting, sticking */
    else
        {
        if(f_abs(fx[n]) >= 1.005 * cof * f_abs(fy[n]))
            cond = 2;                  /* contacting, slipping */
        }
 if(cong)                              /* sticking by operator request */
    cond = 1;

/*                                                                      */
/*  |      Take action based on the condition                    |     */
/*                                                                      */
switch(cond)                           /* Switch based on condition */
   {
   case 0:                             /* not contacting */
     if(f_abs(y1 - y2) < 0.0002 && f_abs(y2 - y3) < 0.0002
       && f_abs(y1 - y3) < 0.0002 && count > 2.0)
       {                               /* check the last three y values */
       crt_srcp(18,60,0);
       printf("Touch down");
       limit = y[n];
       x_target = x[n];                /* desired location of end */
       x2_desired[1] = x[n];           /* fixed location for end */
       x2_desired[2] = y[n];           /* fixed location for end */
       y[0] = y[0] - mmove;            /* move end down */
       drawfloor(limit,1);
       }
     break;
   case 1:                             /* contacting, sticking */
     if(fx[n] <= 0.0)
        x2_desired[1] = x_target + 0.0001;
     if(fx[n] > 0.0)
        x2_desired[1] = x_target - 0.0001;
     crt_srcp(18,60,0);
     printf("sticking           ");
     if(
       ( 0.0 < x[n]-x_target && x[n]-x_target < 0.0002
       && f_abs(y[n]-limit) < 0.0002 && fx[n] < 0.0) ||
                                     /* moving to the left */
       ( 0.0 < x_target-x[n] && x_target-x[n] < 0.0002
       && f_abs(y[n]-limit) < 0.0002 && fx[n] > 0.0) )
                                     /* moving to the right */
        {
        y[0] = y[0] - mmove;         /* move end down */
        crt_srcp(18,60,0);
        printf("stick cong       ");
        }
```

```
      break;
    case 2:                          /* contacting, slipping */
      x2_desired[1] = 0.995 * sgn(fx[n]) * cof * f_abs(fy[n]);
      crt_srcp(18,60,0);
      printf("slipping          ");
      save = (cof * f_abs(fy[n]) - f_abs(fx[n]))/f_abs(fx[n]);
      if( save > 0.0 && save < 0.02 && f_abs(y[n]-limit) < 0.0002 )
         {
         y[0] = y[0] - mmove;        /* move end down */
         crt_srcp(18,60,0);
         printf("slip cong      ");
         /* don't let slip push to beyond where it started */
         if( (x_target > x[n] && fx[n] < 0.0) ||
             (x_target < x[n] && fx[n] > 0.0) )
             x_target = x_target;
         else
             x_target = x[n];
         }
      break;
    }                                /* end of switch condition */


/*                                                              */
/*   What is the condition of the beam?                         */
/*                                                              */
if(limit == 0.0)
    cond = 0;                        /* not contacting */
else
    if(f_abs(fx[n]) < 0.995 * cof * f_abs(fy[n]))
        cond = 1;                    /* contacting, sticking */
    else
        {
        if(f_abs(fx[n]) >= 1.005 * cof * f_abs(fy[n]))
            cond = 2;                /* contacting, slipping */
        }
crt_srcp(14,60,0);
printf("                ");
if(cong)                             /* sticking by operator request */
    {
    cond = 1;
    crt_srcp(14,60,0);
    printf("forced stick");
    }


/*                                                              */
/*   Has the beam moved down too far?                           */
/*                                                              */
   if((y[0] <= limit || y[n-3] <= limit) && limit != 0.0)
```

```
    {
    mmove = 0.0;                    /* exit iteration loop */
    crt_srcp(18,60,0);
    printf("End of Solution");
    }


/*                                                          */
/*  |          Check for keyboard input                  |  */
/*                                                          */
   if(key_scan() == -1)             /* check for a keyboard entry */
     {
     crt_srcp(1,64,0);
     printf("Nk       ");          /* no key pressed */
     }
   else
     {
     d_key = key_getc();            /* get the key from the key board */
     d_key_l = d_key & 0xff;        /* low byte is the ASCII */
     d_key_h = d_key>>8;            /* high byte is the scan code */
     crt_srcp(1,64,0);
     printf("k %d %d",d_key_l,d_key_h); /* key pressed */


     switch(d_key_l)                /* check for ASCII characters */
       {
       case 27 :                    /* check for ESC */
         j = 0;                     /* get out of program, leave while */
         break;
       case 76 :                    /* check for "L" */
       case 108 :                   /* check for "l" */
         crt_srcp(23,60,0);
         printf("Lotus File  ");
         sinput(23,60,80,name2,&arrow);
         crt_srcp(23,60,0);
         printf("%s",name2);        /* make lotus file */
         ptr2 = fopen(name2,"w");   /* write file to disk */
         if(ptr2)                   /* Valid stream ? */
           {
           for(q1=0; q1<=n; ++q1)
              fprintf(ptr2,"%le %le\n",x[q1],y[q1]);
           fclose(ptr2);
           }
         else                       /* not a valid stream */
           {
           crt_srcp(23,60,0);
           printf("File error\007");
           }
         break;
```

```
case 84 :                       /* check for "T" */
case 116 :                      /* check for "t" */
  crt_srcp(23,60,0);
  printf("clip        ");
  clip = finput(23,60,80,clip,&arrow);
  crt_srcp(23,60,0);
  printf("%f",clip);
  break;
case 83 :                       /* check for "S" */
case 115 :                      /* check for "s" */
  if(cong == 0)
     cong = 1;                  /* make the beam stick */
  else
     cong = 0;                  /* back to normal */
  break;
case 88 :                       /* check for "X" */
case 120:                       /* check for "x" */
  crt_srcp(23,60,0);
  printf("%f",x_target);
  x_target = finput(23,60,80,x_target,&arrow);
  crt_srcp(23,60,0);
  printf("%f",x_target);
  break;
case 70 :                       /* check for "F" */
case 102:                       /* check for "f" */
  crt_srcp(23,60,0);
  printf("%f",cof);
  cof = finput(23,60,80,cof,&arrow);
  crt_srcp(23,60,0);
  printf("%f",cof);
  break;
case 77 :                       /* check for "M" */
case 109:                       /* check for "m" */
  crt_srcp(23,60,0);
  printf("%f",mmove);
  mmove = finput(23,60,80,mmove,&arrow);
  crt_srcp(23,60,0);
  printf("%f",mmove);
  break;
case 0:                         /* check for function keys */
  switch(d_key_h)               /* which function key was pressed */
    {
     case 72:                   /* Up arrow key */
      y[0] = y[0]+amove;        /* move end of beam */
      break;
    case 80:                    /* Down arrow key */
      y[0] = y[0]-amove;        /* move end of beam */
      break;
    case 77:                    /* Right arrow key */
      x[0] = x[0]+amove;        /* move end of beam */
```

```
            break;
          case 75:                  /* Left arrow key */
            x[0] = x[0]-amove;      /* move end of beam */
            break;
          case 61:                   /* F3 key */
            print_2 = 1;
            break;
          case 62:                   /* F4  key */
            print_2 = 0;
            break;
          case 63:                   /* F5  key */
            print_1 = 1;
            break;
          case 64:                   /* F6  key */
            print_1 = 0;
            for(q=3; q<=18; q++)
               {
               crt_srcp(q,60,0);
               printf("                    ");
               }
            break;
          default:                   /* do nothing for other keys */
            break;
          }                          /* end of switch d_key_h */
        break;                       /* end of case 0 */
      default:                       /* do nothing for other keys */
        break;

      }                              /* end of switch d_key_1 */
    }                                /* end of key_scan else */


/*                                  */
/*  Print some data                 */
/*                                  */

 if(print_1)                         /* print data */
 {
 crt_srcp(3,60,0);
  printf("%3.4e ",fx[n]);
 crt_srcp(4,60,0);
  printf("%3.4e ",x2_desired[1]);
 crt_srcp(5,60,0);
  printf("%3.4e ",x2_desired[2]);
 crt_srcp(6,60,0);
  printf("%3.4e ",x2_desired[3]);
 crt_srcp(7,60,0);
  printf("%3.4f %3.5f",x[0],x[n]);
 crt_srcp(8,60,0);
  printf("%3.4f %3.4f",y[0],y[n]);
```

```
  crt_srcp(12,60,0);
   printf("%3.4f",fx[n]/fy[n]);
  }


  if(print_2)                          /* print data */
  {
  }


  crt_srcp(19,60,0);                   /* print out the last few values */
   printf("%3.4f",y[n]);               /* of the end deflection */
  crt_srcp(20,60,0);
   printf("%3.4f",y1);
  crt_srcp(21,60,0);
   printf("%3.4f",y2);
  crt_srcp(22,60,0);
   printf("%3.4f",y3);
   y3 = y2;
   y2 = y1;
   y1 = y[n];

}                                      /* close weight loop, while */


/*                                                                    */
/*  End of solving beam equations, print out some data                */
/*                                                                    */

crt_srcp(16,60,0);                     /* print end position */
printf("xend = %f\n",x[n]);
crt_srcp(17,60,0);
printf("yend = %f\n",y[n]);
crt_srcp(18,60,0);
printf("θend = %f\n",th[n]);

crt_srcp(24,60,0);                     /* Continue ? */
printf("hit any key");
junk = key_getc();
if (junk==6400){                       /* print data on printer */
              ptr=fopen("PRN:","w");
              if(ptr){
                     fprintf(ptr,"%f %f %f \n",l,x[n],y[n]);
                     fclose(ptr);
                     }
              else   {
                     printf("trouble\n");
                     fclose(ptr);
                     }
              }
```

```
    if (junk==8704)                    /* draw graph on printer */
       gprint();

    crt_mode(3);                       /* get out of graphic mode */
    closegraphics();                   /* close halo graphics environment */
    print_menu = 1;                    /* Do print out main menu */
    item1 = 0;                         /* go to main menu */
  break;                               /* end of case 4 */


/*                                                                  */
/*  ┌──────────────────────────────┐                                */
/*  │      Exit program to DOS     │                                */
/*  └──────────────────────────────┘                                */

case 5:                                /* do nothing, just exit to DOS - */
  crt_cls();                           /* by exiting the while loop */
  printf("Exitting from BEAM.EXE\n");
  printf("Returning to DOS\n");
  item1 = -1;                          /* leave the the while loop */
  break;                               /* end of case 5 */



default:                               /* default case */
   printf("selection was not 1,2,3,4, or 5 \g\n");
   break;                              /* end of default case */

}                                      /* end of switch(item1) */

}                                      /* close main */


    /*                                                              */
    /*  ┌──────────────────────────────────────────────────────┐    */
    /*  │  SUB-ROUTINES FOR ELASTICA BEAM PROGRAM, CSEGSUB.OBJ │    */
    /*  └──────────────────────────────────────────────────────┘    */


/*                                                                  */
/*  ┌──────────────────────────────────────┐                        */
/*  │  Assemble V Vector  subroutine       │                        */
/*  └──────────────────────────────────────┘                        */

v_vec(fx,fy,m,v)
double fx,fy,m,v[];
   {
   v[1] = fx;
   v[2] = fy;
```

```
    v[3] = m;
    }


/*                                                                    */
/*  |          Assemble F Vector  subroutine                        |  */
/*                                                                    */

f_vec(fx,fy,m,x,y,cond,f,x2_desired)
double fx,fy,m,x,y;                          /* current x2 values */
double f[];                                  /* discrepency vector */
double x2_desired[];                         /* desired boundary conditions */
int cond;                                    /* if 0 beam is not on table */
{                                            /* if 1 beam is on table */
switch(cond)
   {
   case 0:                                   /* not contacting */
     f[1] = fx - x2_desired[1];              /* forces on beam end to be */
     f[2] = fy - x2_desired[2];              /* zeroed */
     f[3] =  m - x2_desired[3];              /* moment to be zeroed */
     break;
   case 1:                                   /* contacting, sticking */
     f[1] = x - x2_desired[1];               /* position of beam end is to */
     f[2] = y - x2_desired[2];               /* be zeroed */
     f[3] = m - x2_desired[3];               /* moment to be zeroed */
     break;
   case 2:                                   /* contacting, slipping */
     f[1] = fx - x2_desired[1];
     f[2] =  y - x2_desired[2];
     f[3] =  m - x2_desired[3];
     break;
   }
}


/*                                                                    */
/*  |       calculate guess of m[0] subroutine                      |  */
/*                                                                    */

double m_guess(b)
double b;                                    /* bending length */
 {
  return(-3.06993e-02 * b +
          4.73752e-04 * pow(b,2.0) +         /* Polynomial curve fit */
          4.82333e-06 * pow(b,3.0) +         /* from previous runs.  */
         -3.24428e-07 * pow(b,4.0) +         /* This insures that the */
          5.77591e-09 * pow(b,5.0) +         /* guess is close to the */
         -5.38162e-11 * pow(b,6.0) +         /* desired solution. */
          2.83040e-13 * pow(b,7.0) +
         -7.95118e-16 * pow(b,8.0) +
```

```
            9.28706e-19 * pow(b,9.0) +
            1.0014244153 );
    }


#include "stdio.h"
#include "math.h"
#define NMAX 5

/*                                                                    */
/*                        Beam plotting function                      */
/*                                                                    */


plot_beam(nn,ln,xn,yn,x_old,y_old)
int nn;                                 /* number of nodes */
double ln;                              /* length of the beam, inches */
double xn[],yn[];                       /* arrays new x-y coordinates */
double x_old[],y_old[];                 /* arrays old x-y coordinates  */

    {
    float xf,yf;                        /* temporary floats */
    int q = 0;                          /* looping variable */
    int gcolor;                         /* graphics color */

    xf = x_old[0];                      /* erase old beam plot */
    yf = y_old[0];
    movabs(&xf,&yf);                    /* move to the first point */
    gcolor = 0;                         /* set to background color */
    setcolor(&gcolor);                  /* set graphics color */
    for(q=1;q<=nn;++q)                  /* erase the old line*/
       {
       xf = x_old[q];
       yf = y_old[q];
       lnabs(&xf,&yf);
       }


    xf = xn[0];                         /* plot new beam plot */
    yf = yn[0];
    movabs(&xf,&yf);                    /* move to the first point */
    gcolor = 3;                         /* set to foreground color */
    setcolor(&gcolor);                  /* set graphics color */
    for(q=1;q<=nn;++q)                  /* draw a line between the nodes */
       {
       xf = xn[q];
       yf = yn[q];
```

```
        lnabs(&xf,&yf);
        }

    }                                   /* close plot_beam() function */

/*  ┌──────────────────────────────────────────────────────────────────┐  */
/*  │     Calculate the sign of a double precision number, function    │  */
/*  └──────────────────────────────────────────────────────────────────┘  */

sgn(f_number)
  double f_number;                      /* floating point number */
    {
    int p = 1;
    if(f_number < 0.0)
       p = -1;
    return(p);
    }

/*  ┌──────────────────────────────────────────────────────────────────┐  */
/*  │                      Menu logic function                         │  */
/*  └──────────────────────────────────────────────────────────────────┘  */


menu_logic(item,max_item,control)
  int item;                             /* current menu position */
  int max_item;                         /* maximum menu positions */
  int control;                          /* function key entered */
    {

    switch(control)                     /* determine which key was pressed */
      {
      case 0:                           /* return key pressed */
      case 6:                           /* right arrow key pressed */
      case 2:                           /* down arrow key pressed */
        ++item;                         /* next menu item */
        break;
      case 4:                           /* left arrow key pressed */
      case 8:                           /* up arrow key pressed */
        --item;                         /* previous menu item */
        break;
      default:
        --item;                         /* previous menu item */
        break;
      }

      if(item < 0)                      /* loop backward */
        item = max_item;
      if(item > max_item)               /* loop forwards */
        item = 0;
      return(item);
```

```
    }                                   /* end of menu logic */

/*
/*  ┌──────────────────────────────────────────────────────────┐
/*  │                Printfl, printf and locate                │
/*  └──────────────────────────────────────────────────────────┘

printfl(row,column,page,f_string)
  int row;
  int column;
  int page;
  char *f_string;
    {
    crt_srcp(row,column,page,0);
    printf(f_string);
    printf("%d",*f_string);
    printf("%d",&f_string);
    }

/*
/*  ┌──────────────────────────────────────────────────────────┐
/*  │                Absolute value of float                   │
/*  └──────────────────────────────────────────────────────────┘

double f_abs(number)
   double(number);
     {
     return(sqrt(number*number));
     }

     /*
     /*  ┌──────────────────────────────────────────────────────┐
     /*  │                Runge-Kutta subroutine                │
     /*  └──────────────────────────────────────────────────────┘

rungekutta(fx,fy,m,th,x,y,ei,ls,w,b,n)
double fx[],fy[],m[],th[],x[],y[];
double ei;                          /* flexural rigidity */
double ls;                          /* segment length */
double w;                           /* unit beam weight, lbm/in» */
double b;                           /* beam width, inches */
int n;                              /* number of nodes */

{                                   /* begin runge-kutta subroutine */
int k,q;                            /* general indexing variable */
double st;                          /* segment length */
double fxt,fyt,mt,tht;              /* temporary variables*/
```

```
double j1,j2,j3,j4;                    /* Runge-Kutta variables */
double k1,k2,k3,k4;                    /* Runge-Kutta variables */
double l1,l2,l3,l4;                    /* Runge-Kutta variables */
double m1,m2,m3,m4;                    /* Runge-Kutta variables */
extern FILE *fopen();                  /* file opening function */
extern int fclose();                   /* file closing function */
FILE *ptr;                             /* stream for file */


        /*      DEBUG */

             ptr=fopen("PRN:","w");
             if(ptr){}
             else    {
                     printf("trouble\n\a");
                     fclose(ptr);
                     }


    for(k = 0; k <= n-1; ++k)
      {
      /* FIRST RUNGE-KUTTA STEP */
      st = k * ls + ls;                /* set segment length temporary */
      fxt = fx[k];                     /* set fx force temporary */
      fyt = fy[k];                     /* set fy force temporary */
      mt = m[k];                       /* set m moment temporary */
      tht = th[k];                     /* set angle θ temporary */

      j1 = ls * (0.0);                 /* d(fx)/ds first RK term */
      k1 = ls * (w * b);               /* d(fy)/ds first RK term */
      l1 = ls * (- fyt * cos(tht) + fxt * sin(tht));
                                       /* d(m)/ds first RK term */
      m1 = ls * (mt/ei);               /* d(θ)/ds first RK term */

      /* SECOND RUNGE-KUTTA STEP */
      st  = k * ls + 0.5 * ls + ls;    /* set segment length temporary */
      fxt = fx[k] + 0.5 * j1;          /* set fx force temporary */
      fyt = fy[k] + 0.5 * k1;          /* set fy force temporary */
      mt  =  m[k] + 0.5 * l1;          /* set m moment temporary */
      tht = th[k] + 0.5 * m1;          /* set angle θ temporary */

      j2 = ls * 0.0;                   /* d(fx)/ds second RK term */
      k2 = ls * (w * b);               /* d(fy)/ds second RK term */
      l2 = ls * (- fyt * cos(tht) + fxt * sin(tht));
                                       /* d(m)/ds second RK term */
      m2 = ls * (mt/ei);               /* d(θ)/ds second RK term */


      /* THIRD RUNGE-KUTTA STEP */
```

```
        st  = k * ls+0.5 * ls + ls;     /* set segment length temporary */
        fxt = fx[k] + 0.5 * j2;         /* set fx force temporary */
        fyt = fy[k] + 0.5 * k2;         /* set fy force temporary */
        mt  =  m[k] + 0.5 * 12;         /* set m moment temporary */
        tht = th[k] + 0.5 * m2;         /* set angle θ temporary */

        j3 = ls * (0.0);                /* d(fx)/ds third RK term */
        k3 = ls * (w * b);              /* d(fy)/ds third RK term */
        l3 = ls * (- fyt * cos(tht) + fxt * sin(tht));
                                        /* d(m)/ds third RK term */
        m3 = ls * (mt/ei);              /* d(θ)/ds third RK term */

        /* FOURTH RUNGE-KUTTA STEP */
        st  = k*ls+ls + ls;             /* set segment length temporary */
        fxt = fx[k]+j3;                 /* set fx force temporary */
        fyt = fy[k]+k3;                 /* set fy force temporary */
        mt  =  m[k]+l3;                 /* set m moment temporary */
        tht = th[k]+m3;                 /* set angle θ temporary */

        j4 = ls * (0.0);                /* d(fx)/ds fourth RK term */
        k4 = ls * (w * b);              /* d(fy)/ds fourth RK term */
        l4 = ls * (- fyt * cos(tht) + fxt * sin(tht));
                                        /* d(m)/ds fourth RK term */
        m4 = ls * (mt/ei);              /* d(θ)/ds fourth RK term */

        /* SUM THE RUNGE-KUTTA TERMS FOR THE NEXT POINT */

        fx[k+1] = fx[k] + (j1 + 2.0* j2 + 2.0* j3 + j4)/6.0;
        fy[k+1] = fy[k] + (k1 + 2.0* k2 + 2.0* k3 + k4)/6.0;
         m[k+1] =  m[k] + (l1 + 2.0* l2 + 2.0* l3 + l4)/6.0;
        th[k+1] = th[k] + (m1 + 2.0* m2 + 2.0* m3 + m4)/6.0;

        x[k+1] = x[k] + ls * cos(th[k]);
        y[k+1] = y[k] + ls * sin(th[k]);
        }                               /* end of runge-kutta */

    fclose(ptr);                        /* DEBUG */

}                                       /* end runge-kutta subroutine */

/*                                                                      */
/*  ┌──────────────────────────────────────────────────────────────┐    */
/*  │            Assemble V Vector  subroutine                     │    */
/*  └──────────────────────────────────────────────────────────────┘    */

v_vector_assemble(fx,fy,m,ei,v)
double fx,fy,m,ei,v[];
   {
   v[1] = fx;
   v[2] = fy;
```

```
   v[3] = m;
   v[4] = ei;
   }

/*                                                                    */
/*          Assemble F Vector  subroutine                             */
/*                                                                    */

f_vector_assemble(fx,fy,m,y,f,x2_desired)
double fx,fy,m,y;                          /* current x2 values */
double f[];                                /* discrepency vector */
double x2_desired[];                       /* desired boundary conditions */
   {
   f[1] = fx - x2_desired[1];
   f[2] = fy - x2_desired[2];
   f[3] =  m - x2_desired[3];
   f[4] =  y - x2_desired[4];
   }


/*
        LUDCMP Subroutine for LU Matrix decomposition                 */
/* Given an N x N matrix A, with physical dimensions NP, this routine replaces it
by the LU decomposition of a rowwise permutation of itself.  A and N are input.  A
is output, arranged as in equation (2.3.14) above, INDX is an output vector which
records the row permutation effected by the partial pivoting, D is output as +/- 1
depending on whether the number of row interchanges was even or odd, respectively.
This routine is used in combination with LUBKSB to solve linear equations or
invert a matrix.  Reference "Numerical Recipes", by William H. Press  */


ludcmp(a,n,np,indx, d)
double a[NMAX][NMAX];                      /* Matrix to be decomposed */
double *d;                                 /* pointer to a double */
int n,np,indx[];

{                                          /* beginning of ludcmp */
double tiny = 1e-20;                       /* a small number */
double vv[NMAX];                           /* stores implicit row scaling */
double aamax, sum, dum;
int i, j, k;                               /* indexing integers */
int imax;

*d=1.0;                                    /* no row interchanges yet */
for(i=1; i<=n; ++i)                        /* loop over rows to get the-- */
   {                                       /* implicit scaling information */
   aamax=0.0;
```

```
   for(j=1; j<=n; ++j)
        {
        if(pow(a[i][j]*a[i][j],0.5) > aamax)
          aamax = pow(a[i][j]*a[i][j],0.5);
        }                                   /* end of j loop */
   if(aamax == 0.0)
     printf("singular matrix");             /* no nonzero largest element */
   vv[i] = 1.0/aamax;                       /* save the scaling */
   }                                        /* end of i loop */


for(j=1; j<=n; ++j)                         /* loop over columns of */
   {                                        /* Crout's Method */
   if(j > 1)
     {
     for(i=1; i<=(j-1); ++i)                /* equ. 2.3.12 except for i = j */
        {
        sum = a[i][j];
        if(i > 1)
          {
          for(k=1; k<=i-1; ++k)
             {
             sum = sum - a[i][k] * a[k][j];
             }
          a[i][j] = sum;
          }                                 /* end of if i > 1 */
        }                                   /* end of i loop */
     }                                      /* end of if j > 1 */
   aamax=0.0;                               /* initialize search for */
                                            /* the largest pivot */
   for(i=j; i<=n; ++i)                      /* equa. 2.3.12 for i = j --*/
      {                                     /* and 2.3.13 for i=j+1..N */
      sum = a[i][j];
      if(j > 1)
        {
        for(k=1; k <= (j-1) ; ++k)
           {
           sum = sum - a[i][k] * a[k][j];
           }
        a[i][j] = sum;
        }                                   /* end of if j > 1 */
      dum=vv[i] * fabs(sum);                /* figure of merit for pivot */
      if(dum > aamax)                       /* is it better than the --*/
        {                                   /* best so far */
        imax=i;
        aamax=dum;
        }
      }                                     /* end of j loop */
   if(j != imax)                            /* do we need to inchange rows */
     {                                      /* yes, do so */
```

```
      for(k=1; k<=n; ++k)
         {
         dum = a[imax][k];
         a[imax][k] = a[j][k];
         a[j][k] = dum;
         }
      *d = -1 * *d;                          /* change the parity of d */
      vv[imax] = vv[j];                      /* also interchange the scale */
      }                                      /* end of if j != imax */

    indx[j] = imax;
    if(j != n)                               /* Now, finially, divide by -- */
         {                                   /* the pivot element */
         if(a[j][j] == 0.0)
           {                                 /* if pivot element is zero --*/
           a[j][j] = tiny;                   /* substitute TINY */
           }
         dum = 1.0/a[j][j];
         for(i=(j+1); i<=n; ++i)
            {
            a[i][j] = a[i][j] * dum;
            }
          }                                  /* end of if j != n */


    }                                        /* go back for next column-- */
                                             /* reduction */
if(a[n][n] == 0.0)
    {
    a[n][n] = tiny;
    }
return;

}                                            /* end of ludcmp */
```

```
/*┌──────────────────────────────────────────────────────┐
  │     LUBKSB Subroutine for Solving LU decomposed Matrixes     │*/
└──────────────────────────────────────────────────────┘
/* Solves the set of N linear equations A·X = B.  Here A is input, not
as the matrix A but rather as its LU decomposition, determined by the
routine LUDCMP.  INDX is input as the permutation vector returned by LUDCMP.  B is
input as the right-hand side vector B, and returns with
the solution vector X.  A, N, NP and INDX are not modified by this
routine and can be left in place for successive calls with different
right-hand sides B.  This routine takes into account the possibility
that B will begin with many zero elements, so it is efficient for use
in matrix inversion. Reference "Numerical Recipes", by William H. Press */
```

```
lubksb(a,n,np,indx,b)
double a[NMAX][NMAX];                    /* LU matrix of A */
double b[];                              /* right hand side vector */
int n,np,indx[];

{

                                         /* When ii is set to a positive
                                         value, it will become the index
                                          of the first nonvanishing element
                                          of B.  We now do the forward
                                          substitution, equation 2.3.6.
                                           The only new wrinkle is to
                                          unscramble the permutation
                                          as we go. */
int ii = 0;
int i,j,k,l,ll;
double sum;
int i2,j2;

for(i=1; i<=n; ++i)
   {
   ll = indx[i];
   sum = b[ll];
   b[ll] = b[i];
   if(ii != 0)
       {
       for(j=ii; j<=(i-1); ++j)
            {
            sum = sum - a[i][j]*b[j];
            }
        }
   else
      if(sum != 0.0)
         {                               /* A non zero element was-- */
         ii=i;                           /* encountered, so form now on-- */
         }                               /* do the sums in the loop above */
   b[i] = sum;
   }                                     /* end of i loop */
for(i=n; i>=1; --i)                      /* Now do the backsubstitution,-- */
   {                                     /* equation 2.3.7 */
   sum=b[i];
   if(i < n)
      {
      for(j=i+1; j<=n; ++j)
         {
         sum = sum - a[i][j] * b[j];
         }
      }
   b[i] = sum/a[i][i];                   /* Store a component of X */
```

```
    }
return;
}                                   /* end of lubksb */
```